UNIVERSIDAD DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER UNIVERSITARIO

INGENIERO DE TELECOMUNICACIÓN

Mecanismos de análisis *BigData* para la caracterización de la actividad docente en un Campus Virtual *Moodle*

Autor:
Dña. Alejandra Roldán Mínguez

Tutor:
D. Juan Pablo de Castro Fernández

Valladolid, 21 de abril de 2016

TÍTULO:	**Mecanismos de análisis *BigData* para la caracterización de la actividad docente en un Campus Virtual *Moodle***
AUTOR:	**Dña. Alejandra Roldán Mínguez**
TUTOR:	**D. Juan Pablo de Castro Fernández**
DEPARTAMENTO:	**Departamento de Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**D. Juan Ignacio Asensio**
VOCAL:	**Dña. María Jesús Verdú Pérez**
SECRETARIO:	**Dña. Luisa María Regueras Santos**

FECHA:	**29 de Abril de 2016**
CALIFICACIÓN:	

RESUMEN

El trabajo que aquí se presenta tiene por objetivo el desarrollo de un conjunto de mecanismos o algoritmos que permitan realizar un análisis *BigData* de la información que se genera en un entorno *Moodle*, como el del Campus Virtual de la Universidad de Valladolid. Hoy en día existen multitud de análisis sobre el uso de plataformas similares realizados por medio de técnicas de la Minería de Datos. En este trabajo nos planteamos trasladar estos análisis a un entorno en el que se generan cantidades enormes de datos (como podría ser en el caso de la UVa), y por lo tanto pueda dar sentido al uso de técnicas *BigData*. Para todo ello, se hace uso de la plataforma *open source Apache Hadoop* proporcionada por *Hortonworks*. A través de ella se realizarán y ejecutarán una serie de algoritmos de análisis que permitan extraer conclusiones sobre cómo los usuarios utilizan la plataforma *Moodle* e indicadores sobre la influencia que las actividades, contenidos o demás elementos que a través de ella se proporcionan a los usuarios, tienen sobre los resultados académicos, así como otros resultados de interés que puedan servir para mejorar el uso de la plataforma educativa.

ABSTRACT

The objective of the work that it's presented in this document is to develop a set of mechanisms or algorithms to perform a *BigData* analysis of information generated in a *Moodle* platform, such as the Virtual Campus of the University of Valladolid (UVa). Nowadays there are many studies about the use of similar platforms, using Data Mining techniques. In this work we focus on adapting these analyzes to an environment in which large amount of data are generated (as might be the case of the UVa), and it may be possible to use *BigData* techniques. In this way, *Hortonworks Apache Hadoop* open source platform is used. This *BigData* platform will allow us to define and implement analysis algorithms to extract conclusions about how users use a *Moodle* platform and indicators about the influence that activities, contents and others elements published in *Moodle* have in academic results, as well as other interesting results that may be useful to improve the use of the educational platform.

PALABRAS CLAVE

BigData, *Moodle*, análisis, registros, actividad, usuarios, influencia, correlacion, *Machine Learning*, *Hadoop*, *Hive*, *Sqoop*, *Mahout*, *Oozie*

KEYWORDS

BigData, *Moodle*, analysis, records, activity, users, influence, correlation, *Machine Learning*, *Hadoop*, *Hive*, *Sqoop*, *Mahout*, *Oozie*

Contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Motivación y Objetivos

El siguiente Trabajo Fin de Máster en Ingeniería de Telecomunicación se centra en torno a dos conceptos: *BigData* y *Moodle*. En él se desarrollan una serie de componentes *software* con los que poder analizar el uso que los alumnos hacen de la plataforma *Moodle* de la Universidad de Valladolid (UVa).

El objetivo principal consiste en aplicar los conceptos de *BigData* al análisis estadístico de los ficheros de *log* que registran la actividad de alumnos y profesores (usuarios en general) sobre la plataforma *Moodle*. Estos análisis deberán ser tales que permitan extraer conclusiones útiles para poder mejorar el uso de la plataforma como ayuda educativa. Métricas interesantes serán:

- ¿Acceden los alumnos al material de apoyo más allá de los apuntes de clase?
- ¿Influye el acceso al material de apoyo en la nota final del alumno?
- ¿Acceden los alumnos al material en fechas cercanas a su publicación o más cercanas a la fecha de evaluación?
- Control del acceso de los alumnos a las páginas de los cursos con el objetivo de averiguar qué alumnos han abandonado la asignatura por no acceder a la plataforma en un periodo de tiempo.
- ¿Los alumnos acceden más a la plataforma desde dentro o fuera del centro?
- ¿Leen los alumnos los mensajes que los profesores publican en los foros?

Principalmente se buscarán métricas que permitan que los profesores puedan hacer un seguimiento de la actividad de sus alumnos (según cursos) y poder concluir si están procediendo de forma adecuada en el uso de la plataforma y si consiguen fomentar el uso de la plataforma a sus alumnos.

Debido a que también se registra la actividad de los profesores, podría ser interesante generar un análisis relativo a los profesores para controlar que ellos también hacen un uso adecuado de la plataforma:

- ¿Acceden los profesores de forma regular a sus cursos?

- ¿Hacen uso de la plataforma como medio de comunicación con sus alumnos o tienen su curso completamente abandonado?

De esta forma, el objetivo final será la implementación de una infraestructura *BigData* sobre la que definir unos mecanismos de análisis de *Moodle*, cuyos resultados después puedan ser utilizados para extraer las conclusiones que sean oportunas y que permitan responder a las cuestiones anteriores.

1.2. *BigData*

1.2.1. ¿En qué consiste el término *BigData*?

Definir el término *BigData* (comúnmente denominado Datos Masivos en español) suele dar lugar a confusión, ya que en muchas ocasiones se suele adaptar a la situación concreta en la que se vaya a usar la infraestructura correspondiente. Se habla de *BigData* cuando se necesita una solución para hacer frente a grandes cantidades de datos, tales que no se pueden procesar o almacenar en una infraestructura común. Este es un concepto que ha crecido enormemente en los últimos años, para usos muy distintos, y por tanto ha dado lugar a la aparición de multitud de definiciones distintas. Por ejemplo, [1] muestra en su artículo un gráfico (adaptado en la Figura 1) con las distintas respuestas sobre la definición de *BigData* obtenidas al cuestionar a distintos ejecutivos del sector.

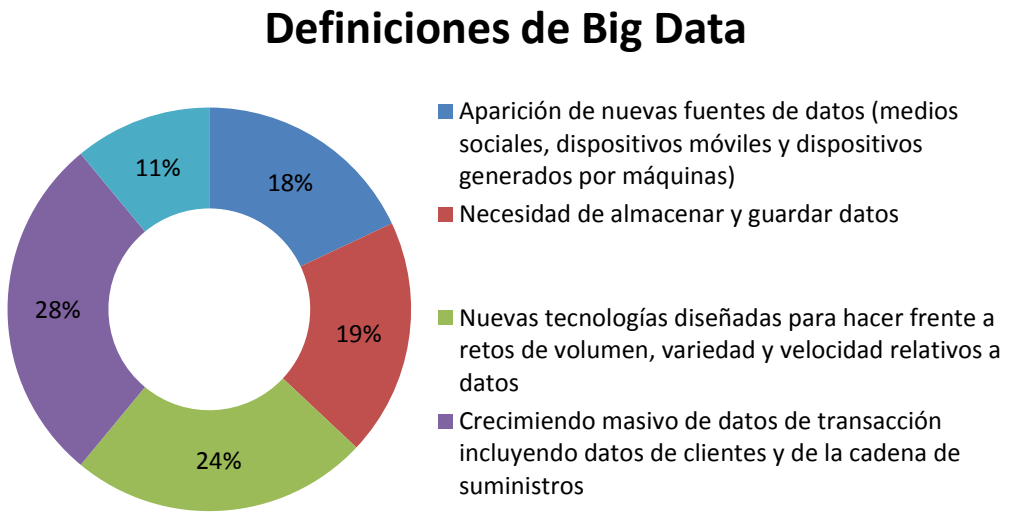


Figura 1. Definiciones de *BigData* según una encuesta en línea a 154 ejecutivos en abril de 2012. Fuente: Adaptado de [1]

A pesar de que la característica fundamental del *BigData* es la gran cantidad de datos con los que debe tratar, muchos autores, como el anteriormente mencionado, indican que este no es el único aspecto que se debe tener en cuenta a la hora de definir o tratar con *BigData*. En este sentido, se habla de las Tres V's (*Three* V's): Volumen (*Volume*), Variedad (*Variety*) y Velocidad (*Velocity*). A estas se suelen añadir como características complementarias, otras tres V's : Veracidad (*Veracity*), Variabilidad (*Variability*) y Valor (*Value*).

- El **volumen** se refiere a la gran cantidad de datos generados en muy poco tiempo. Normalmente la gran cantidad de datos a tratar se extiende hasta el orden de *terabytes* o *petabytes* de datos. Algo imposible, por ejemplo, de almacenar en un dispositivo común. Como ejemplo real, se estima [2] que *Facebook* tiene que tratar el procesamiento

de más de un millón de fotografías por segundo, lo que da lugar a alrededor de 60 *gigabytes* de datos.

- La información que se puede gestionar es muy heterogénea, lo que da lugar a una gran **variedad** de contenidos. Principalmente estos datos se clasifican en tres grandes grupos: estructurados (como los que se almacenan en bases de datos o las hojas de cálculo), semi-estructurados (como aquellos en formato HTML, XML o JSON) o no estructurados (como PDFs o email). Se estima [1] que aproximadamente el 95 % de los datos almacenados en un contexto *BigData* son de tipo no estructurado. Por supuesto, el hecho de poder encontrarnos gran cantidad de distintos tipos, formatos (audio, texto, video, etc.) y orígenes (sensores, redes sociales, etc.) de datos da lugar a diferentes métodos de análisis y procesamiento de la correspondiente información.

- Con **velocidad** nos referimos a la tasa a la que los datos son generados y con qué rapidez estos datos deben ser analizados y tomados en cuenta para acciones consecuentes. En algunos casos se llegan a necesitar infraestructuras de análisis con capacidad para realizar cálculos y procesamientos en tiempo real, por ejemplo, para generar alarmas ante un determinado comportamiento que está sucediendo en un instante concreto.

- El concepto de **veracidad** hace referencia al hecho de que en ocasiones se trabaja con datos procedentes de fuentes de datos cuya fiabilidad no es conocida o directamente no son fiables, por ejemplo, cuando se analizan datos procedentes de redes sociales, marcados por la subjetividad del autor.

- La **variabilidad** suele ir ligada a otro concepto, la complejidad. De naturaleza más técnica que la característica anterior, se refiere a la variación de las tasas de flujo de datos que se experimentan en una infraestructura *BigData*. Por otro lado, el manejar datos que puedan proceder de multitud de fuentes externas distintas, da lugar a una complejidad a nivel de conexión, concordancia y transformación de datos.

- El **valor** de los datos *BigData* se considera bajo en relación con su volumen, es decir, se manejan cantidades enormes de datos, pero cada uno de estos datos prácticamente no tiene valor por sí mismo.

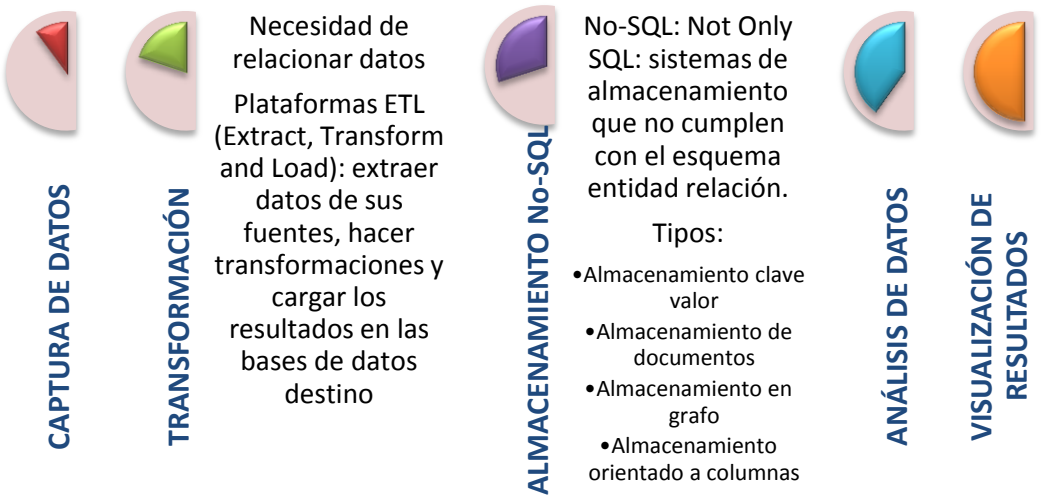


Figura 2. Procedimiento típico en un problema *BigData*

A pesar de existir distintas alternativas para hacer frente a un problema de *BigData*, como se verá en el Capítulo 2, existe una metodología básica para proceder, tal y como se esquematiza en la Figura 2.

1.2.2. Ejemplos de uso de *BigData*

Debido al enorme auge que los sistemas informáticos y la tecnología en general han tenido en los últimos años, son numerosas las compañías u organizaciones que han tenido que trasladar sus, en un principio, pequeños problemas de cálculo, procesamiento y almacenamiento de datos a un problema *BigData*. En esta sección se muestra algún ejemplo representativo de uso de *BigData* donde se podrá apreciar la gran disparidad de fines de esta utilización:

- *Facebook* gestiona más de 50 mill millones de fotos de sus usuarios [3].
- *eBay.com* gestiona búsquedas, recomendaciones de clientes y *merchandising* [4].
- *Amazon* hace uso de *BigData* para gestionar millones de operaciones genéricas de datos por día [5].
- *Google* gestiona 100 mill millones de búsquedas al mes [6].
- La *NASA* posee un centro de simulación de clima (*NASA Center for Climate Simulation*, *NCCS*) donde almacena cantidades de observaciones climáticas y datos de simulación [7].

1.3. Plataforma *Moodle*

La plataforma *Moodle* [8] (*Modular Object Oriented Dynamic Learning Environment*) es un sistema de gestión de cursos gratuito en forma de aplicación *web* orientado al aprendizaje *online*, lo que técnicamente se conoce como Sistema de Gestión de Contenidos Educativos (*Learning Content Management System, LCMS*).

Este paquete *software* se comercializa bajo licencia *Open Source*, lo que implica un acceso libre al mismo. Es por ello que multitud de organizaciones (principalmente educativas) e individuales contribuyen a su mejora y evolución. Son muchas las características por las que esta plataforma ha desbancado a cualquier otra alternativa. Entre ellas se pueden destacar:

- Facilidad de uso, de instalación y de administración.
- Totalmente gratuito.
- Actualización constante para adecuarse a las necesidades cambiantes de los usuarios y a la evolución tecnológica.
- Disponible en más de 120 idiomas.
- Plataforma de aprendizaje todo en uno ya que soporta tanto aprendizaje mixto (*blended learning*) como totalmente en línea. Además, cuenta con herramientas integradas como foros, *wikis*, *chats* y *blogs*.

- Flexible y personalizable, ya que es un proyecto de código abierto.
- Escalable a cualquier cantidad de datos.
- Robusto, seguro y privado.
- Accesible desde cualquier dispositivo y cualquier lugar.
- Soportado por una enorme comunidad internacional de personas que colaboran altruistamente con el proyecto y un equipo de desarrolladores dedicados exclusivamente a *Moodle*.

En el momento de realización de este trabajo, *Moodle* gestionaba más de 8.6 millones de cursos en un total de 222 países. Más estadísticas pueden consultarse en [9].

Para la realización de este trabajo, la parte interesante de *Moodle* es el registro de la actividad de los usuarios de *Moodle* (alumnos y profesores). Por lo general, por cada acción llevada a cabo dentro de *Moodle* se guardan cuatro datos [10]: quién (*who*), qué (*what*), cuándo (*when*) y dónde (*where*); o lo que es lo mismo: qué usuario inicia la acción, desde dónde, cuándo la inicia y cuándo abandona la sesión en *Moodle*. A partir de esta información se pueden realizar multitud de análisis distintos.

Tras esta pequeña explicación acerca del formato de los registros de *Moodle* se puede apreciar ciertos aspectos característicos de una infraestructura *BigData*, principalmente los siguientes:

- En primer lugar, una única entrada no tiene ningún valor por sí misma, tiene que usarse en relación con otras tantas entradas con las que pueda guardar relación, por ejemplo, todos las entradas relativas a un alumno concreto para analizar cuantas veces accede a *Moodle* al día.
- Por otro lado, es obvio que a lo largo de un único día se pueden obtener miles de registros, ya que la información se almacena con un enorme detalle, lo que da lugar a que un acceso simple de un usuario a su página de *Moodle* dé lugar a varias acciones: *login*, acceso a página principal, acceso a página de un recurso, vista del recurso, vuelta a la página principal, etc. El registro de la información de actividad de profesores y alumnos a lo largo de la duración de un curso generará una gran cantidad de datos.

Todo ello pone de manifiesto el interés de estudiar las posibilidades de *BigData* para poder realizar un análisis de los registros de la actividad en *Moodle,* como se pretende realizar en este trabajo.

1.4. Estructura de la memoria

El contenido del resto del documento se estructura como sigue. En el Capítulo 2 se presenta el estado del arte relacionado con el trabajo a realizar. En concreto, se analizan, por un lado el uso de la Minería de Datos para el análisis de plataformas educativas como *Moodle*; y por otro lado, la plataforma *BigData* utilizada para la realización de este trabajo, junto con conceptos relacionados con este tipo de procesamiento de datos.

En el Capítulo 3 se analiza el problema a resolver en este trabajo, cuyo desarrollo se detalla en el Capítulo 4. Las conclusiones al mismo se presentan en el Capítulo 5 mientras que el Capítulo 6 presenta trabajos futuros relacionados con el que aquí se presenta. Finalmente, el documento se completa con Referencias y Contenido del CD adjunto.

Capítulo 2

Estado del arte

A la hora de afrontar el trabajo previamente enumerado nos podemos plantear dos grandes cuestiones:

- ¿Qué bases, procedimientos o técnicas existen dentro del *BigData* para afrontar nuestro problema?
- ¿Qué otros estudios se han llevado a cabo en el campo del análisis de la actividad en *Moodle*?

La primera pregunta tiene por objetivo encontrar algún método o alguna infraestructura que podamos usar para desarrollar nuestro análisis *BigData* sin tener que partir desde cero, usando arquitecturas probadas y utilizadas en contextos que puedan ser similares al nuestro. Respuestas a la segunda pregunta nos pueden ayudar a entender en qué punto está hoy en día el análisis de los registros de actividad en *Moodle*. De esta forma podremos saber si algunos de estos estudios pueden ser aplicables o adaptados a nuestro caso o incluso comprobar si podemos aportar algo novedoso que todavía no se haya desarrollado.

Con estos objetivos en mente, a lo largo de este capítulo se desarrollarán varios temas. En primer lugar, se hará un breve repaso a los conceptos de *Data Mining* y *Machine Learning* enumerando los algoritmos más básicos que se pueden usar para analizar información. A continuación, nos centraremos en analizar otros estudios que relacionan *Data Mining*, es decir, análisis de datos, y *Moodle*.

En la segunda sección nos adentraremos en el mundo *BigData* para buscar qué alternativas tenemos para poder implementar, de la forma más fácil y rápida, un entorno de trabajo para intentar adaptar algunos de los estudios del apartado 2.1.2 usando técnicas *BigData*. Para ello seguiremos tres pasos, en primer lugar trataremos de buscar, lo que se denomina, paradigmas de programación que se pueden usar para realizar procesamiento de datos. A continuación, nos centraremos en plataformas *open source* que usan el paradigma *MapReduce* como base y algunas adaptaciones o extensiones de este paradigma con el objetivo de mejorar su rendimiento. Obviamente, de entre todos los paradigmas que se estudien, se ha optado por escoger *MapReduce*, dada su enorme implantación, no tanto del propio paradigma, sino de una de sus implementaciones más conocidas, *Apache Hadoop*. Esta será la plataforma escogida para desarrollar nuestro trabajo. Es por este motivo, que la siguiente sección se encargará de analizarla, averiguar qué nos puede ofrecer para realizar nuestro trabajo y cómo podemos acceder a ella.

2.1. *Data Mining* y plataformas educativas

Este primer apartado se centrará en revisar, primero, las técnicas de *Data Mining* más extendidas y, segundo, cómo se realiza hoy en día el análisis del uso de plataformas *e – learning* en general, y *Moodle* en particular, mediante dichas técnicas.

2.1.1. *Data Mining* & *Machine Learning*

La Minería de Datos (*Data Mining*, DM) y el Aprendizaje Automático (*Machine Learning*, ML) son dos conceptos muy relacionados, pero, aunque en ocasiones se usan indistintamente, existen diferencias en su definición.

Machine Learning se refiere al estudio, diseño y desarrollo de algoritmos que proporcionan a los ordenadores la capacidad de aprender sin necesidad de haber sido explícitamente programados [11]. Por otro lado, ***Data Mining***, también denominada *Knowledge Discovery in Databases* (*KDD*, Descubrimiento de conocimiento en bases de datos) se define como el proceso de extraer conocimiento oculto y desconocido, pero de gran interés, de grandes cantidades de datos [12]. Para ello, hace uso de algoritmos de *Machine Learning*.

A pesar de que cada estudio concreto puede ser completamente distinto a los demás, por lo general, se suelen distinguir cuatro **etapas en un proceso *Data Mining*** [13]:

- **Recolección de datos**: en primer lugar, se obtienen de la base de datos la información sobre la que realizar el análisis.

- **Pre – procesamiento de los datos**: en ocasiones, los datos brutos no se pueden utilizar, por ello, en esta segunda etapa los datos se filtran, se eliminan aquellos que no son de utilidad para el estudio o se transforman los datos originales para un mejor procesamiento. Por ejemplo, es bastante común transformar los valores numéricos a rangos discretos, por ejemplo, si el valor está entre 0 y 5 se transforma a BAJO y si está entre 5.1 y 10, se transforma a ALTO. El pre – procesamiento necesario dependerá de los datos que tengamos y lo que necesitemos hacer con ellos, así como los requerimientos del algoritmo utilizado.

- **Aplicación de los algoritmos**: a continuación, se aplica el algoritmo de *Machine Learning* correspondiente, en función de lo que queramos realizar.

- **Interpretación, evaluación y obtención de los resultados**: por último, se analizan los resultados obtenidos para extraer esa información útil que en los datos originales estaba oculta.

El resto del apartado puede resultar en ocasiones redundante, dada la cercanía entre la minería de datos y los mecanismos de aprendizaje automático.

Se pueden distinguir hasta diez **tareas** distintas a llevar a cabo en **Minería de Datos** [14], [15]:

- **Detección de anomalías** (*anomaly detection*). Identificación de datos inusuales dentro del conjunto total de datos considerados.

- **Minería de reglas de asociación** (*association rule mining*). Buscar relaciones entre variables características de los datos.

- **Minería de secuencias de patrones.** Descubrir secuencias de patrones en los datos.
- ***Clustering* o agrupamiento**. Descubrir grupos de datos que son similares con respecto a algún criterio, sin usar ningún patrón origen o criterios de partida.
- **Clasificación** (*classification*). En base a datos anteriores se genera un mecanismo para clasificar en categorías pre – generadas nuevos elementos.
- **Regresión** (*regression*). Proporciona una función que modela los datos con el mínimo error.
- **Resumen** (*summarization*). Proporciona una representación más compacta de los datos.
- **Predicción.** Permite predecir el valor de una variable en base a los valores concretos que toman otras y tomando como referencia anteriores relaciones entre variables de entrada y salida.
- **Minería de texto (*Text Mining*).** Subcategoría de la minería de datos para analizar textos, especialmente el contenido de las páginas *web*, pudiendo analizar datos tanto no estructurados como semi – estructurados, documentos de texto completo, ficheros *HTML* o *emails*.
- **Análisis de redes sociales.** Este es un campo emergente que tiene por objetivo analizar las redes sociales y el uso que de ellas se hace para extraer información de utilidad, tanto de forma genérica como en un contexto más concreto, por ejemplo, tantear las emociones de los trabajadores de una empresa para conocer cuál es su grado de satisfacción.

Para llevar a cabo cualquiera de estas tareas, se usan algoritmos de *Machine Learning*. En primer lugar, estos algoritmos pueden clasificarse según el tipo de aprendizaje [16]:

- **Aprendizaje Supervisado** (*supervised learning*): se trabaja con una serie de datos de partida que se denominan datos de entrenamiento, que permiten caracterizar un conjunto de clases definidas. El correspondiente algoritmo analizará estos datos de entrenamiento para producir una función que se usará para mapear nuevos datos. Dentro de esta categoría se encuadran algoritmos como *SVMs* (*Support Vector Machines*) o *Bayes* Ingenuo.
- **Aprendizaje No Supervisado** (u*nsupervised learning*): en este caso no se tienen datos de entrenamiento ni clases conocidas. Se trata de caracterizar nuevas clases. El correspondiente modelo se va ajustando a las observaciones pero nunca a datos de prueba. Dentro de esta categoría se encuadran algoritmos como las K – Medias o *clustering* jerárquico.

En función de su objetivo o funcionamiento, se pueden distinguir varias categorías de algoritmos, prácticamente una clasificación análoga a las tareas de *Data Mining*, ya que, por ejemplo, los algoritmos de clasificación de *Machine Learning* permitirán llevar a cabo tareas de clasificación *Data Mining*. A continuación de enumeran algunos de los algoritmos más comunes.

Algoritmos de recomendación [16]. Este tipo de algoritmos proporcionan como resultado recomendaciones en base a información de usuario sobre comportamientos anteriores. Por ejemplo, *Amazon* lo usa para recomendar a los usuarios nuevos artículos en base a compras anteriores, o *Facebook* lo usa para recomendar nuevos amigos en base a los que ya se tiene

agregados. A este tipo de algoritmos también se les suele denominar **Algoritmos de Filtrado Colaborativo** (*Collaborative Filtering*) y se les puede considerar bastante parecidos a los Algoritmos de Reglas de Asociación. De hecho, dependiendo de caso, los algoritmos o sistemas de recomendación están implementados con, entre otros, algoritmos de reglas de asociación.

Algoritmos de Reglas de Asociación (*Association Rules*). Las reglas de asociación describen relaciones entre atributos de un conjunto de datos que superan unos determinados umbrales [17]. Las reglas de asociación [15] permiten descubrir relaciones entre atributos de los datos analizados, produciendo reglas *if-then* del tipo *X => Y*, con las que se indica que tras analizar los datos se ha comprobado que es bastante común que cuando en un dato se da el atributo X (antecedente) también se suele dar Y (consecuente), la proporción o probabilidad de esta ocurrencia se puede fijar como valor umbral, de tal forma que solo se consideran útiles aquellas reglas cuyas características están por encima de esos umbrales. Se pueden utilizar hasta cuatro métricas (se denomina transacción a cada una de los registros de los datos origen) [18]:

- ***Support*** o ***soporte****:* se define como la proporción de transacciones que contienen el evento X, es decir, porcentaje de transacciones que contienen tanto X como antecedente e Y como consecuente con respecto al total de transacciones.

$$supp(X) = \frac{número\ de\ transacciones\ que\ contienen\ X}{número\ total\ de\ transacciones}$$

$$supp(X => Y) = \frac{número\ de\ transacciones\ que\ contienen\ X\ e\ Y}{número\ total\ de\ transacciones}$$

- ***Confidence*** o ***confianza***: porcentaje de transacciones que contienen el consecuente Y con respecto a todas las transacciones que contienen X como antecedente.

$$conf(X => Y) = \frac{supp(X \cup Y)}{supp(X)}$$

- ***Lift*** es el ratio del valor de *support* esperado si X e Y fueran independientes.

$$lift(X => Y) = \frac{supp(X \cup Y)}{supp(X) \cdot supp(Y)}$$

- ***Conviction*** se interpreta como el ratio de la frecuencia esperada de que ocurra X sin que ocurra Y.

$$conv(X => Y) = \frac{1 - supp(Y)}{1 - conf(X => Y)}$$

Principalmente se usan las dos primeras para evaluar la "calidad" de la regla. Además, se suelen tener en cuenta simultáneamente varias de estas medidas, estableciendo umbrales para cada una de ellas. Por ejemplo, el soporte nos permitiría detectar aquellas reglas que, aunque se cumplen en algún caso y puedan tener alta confianza, no son relevantes porque cubren casos raros o poco frecuentes (y tienen bajo soporte, por lo tanto).

Las Reglas de Asociación se pueden considerar similares a los sistemas de recomendación ya que se pueden usar con el mismo propósito. Por ejemplo, dada una lista de artículos comprados por un cliente, se pueden obtener reglas que indiquen que cuando un cliente compra un artículo X es común que compre también el artículo Y, por lo que se puede usar este resultado para recomendar a un cliente que ya ha comprado X que también compre Y. Puede verse un ejemplo en la Figura 3. En este ejemplo, se analizan los artículos comprados por cinco clien-

tes. En base a estas compras, se han obtenido cinco reglas de asociación, la primera de ellas dice que con un soporte de 0.4 y una confianza de 0.66 si un cliente compra el artículo A también comprará el artículo D, y así sucesivamente con el resto de reglas.

Rule	Support	Confidence	Lift
$A \Rightarrow D$	2/5	2/3	10/9
$C \Rightarrow A$	2/5	2/4	5/6
$A \Rightarrow C$	2/5	2/3	5/6
$B \& C \Rightarrow D$	1/5	1/3	5/9

Figura 3. Ejemplo sencillo de implementación de reglas de asociación. Fuente: [19]

Existen distintos **algoritmos de reglas de asociación**, pero son dos los más conocidos:

- ***Apriori*** [20]: este es el algoritmo de reglas de asociación más conocido y más utilizado, como se podrá ver en el apartado 2.1.2. Se encarga de identificar la frecuencia con la que elementos individuales (parejas atributo – valor) aparecen en los datos totales y después lo extiende a conjuntos de elementos para encontrar aquellos conjuntos que son muy usuales. Es esta frecuencia de aparición de conjuntos de elementos la que se usa para construir las reglas de asociación más comunes de la base de datos.

- ***Frequent Pattern, FP – Growth*** [21] se encargará de analizar elementos en un grupo e identificar qué elementos típicamente aparecen juntos. No es quizá la técnica de reglas de asociación más utilizada, pero es la única presente en *Apache Mahout* (ver apartado 2.2.3.2.5 para entender porque esto es importante). Para generar las reglas, en primer lugar, cuenta el número de ocurrencias de cada pareja atributo – valor en la base de datos, como en el algoritmo *Apriori*. La diferencia está en el segundo paso: con estos datos genera un árbol en el que ordena los conjuntos de parejas atributo – valor de forma descendente atendiendo a su frecuencia de aparición.

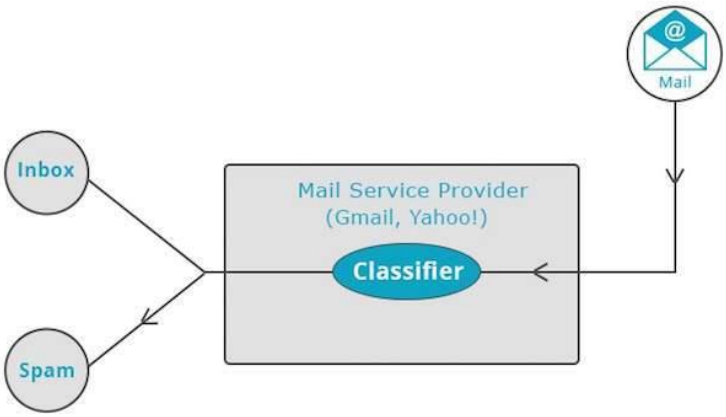


Figura 4. Ejemplo de aplicación de clasificadores. Fuente: [16]

Algoritmos de clasificación. Esta es una técnica que usa los datos conocidos para determinar cómo se deben clasificar los nuevos datos en una serie de categorías existentes, realizando un mapeado de un espacio discreto o continuo a uno discreto, donde cada elemento se

etiquetará como perteneciente a una o varias categorías, dependiendo del tipo de algoritmo [15]. Los algoritmos de clasificación son de tipo supervisado, ya que usan una serie de datos de entrenamiento en base a los cuales generan las clases o etiquetas con las que categorizar nuevos datos. Por ejemplo, *iTunes* usa clasificadores para generar *playlist*, o gestores de correo electrónico como *Yahoo!* o *Gmail* lo usan para separar el correo normal del *spam* (Figura 4).

Cuando se clasifica un conjunto de datos, el sistema clasificador lleva a cabo las siguientes acciones (Figura 5): en primer lugar se preparan los datos de entrenamiento con los que se crea un modelo de datos en el cual se define qué deben cumplir los datos para hacerlos pertenecer a una categoría. Cuando se obtienen los datos de observaciones reales, se aplica ese modelo de datos para clasificarlos.

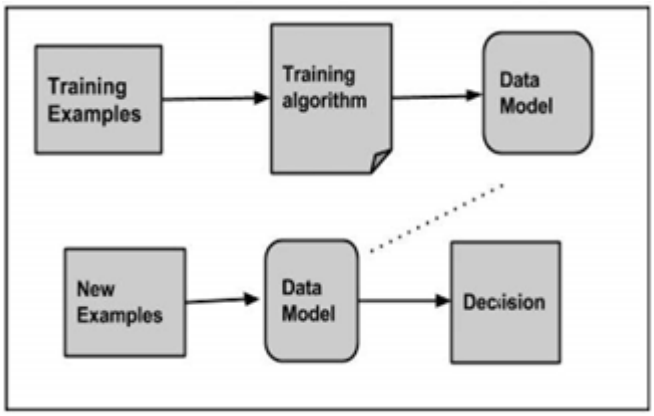


Figura 5. Pasos de un algoritmo de clasificación. Fuente: [16]

Existen multitud de algoritmos de clasificación incluso subcategorías de algoritmos de clasificación. En función del tipo de clases en que clasifican los datos, se distinguen algoritmos de clasificación binaria (solo dos clases) y algoritmos de clasificación multi – clase (más de dos clases) [22]. Por otro lado, la clasificación puede ser de etiqueta única (cada dato solo puede pertenecer a una clase) o multi – etiqueta (un dato puede pertenecer a una o más de una clase simultáneamente). A continuación se enumera alguno de estos algoritmos:

- Las **reglas de asociación** se pueden llegar a considerar algoritmos de clasificación dependiendo del objetivo de las reglas generadas.

- **Árboles de decisión** (*decision trees*). Los algoritmos basados en árboles de decisión [23], [24] tienen por objetivo crear un modelo con el que poder predecir el valor de una variable basándose en otra serie de variables de entrada. Para ello, el conjunto de condiciones que deciden el flujo de la predicción se organiza de forma jerárquica, donde la transición de un nodo a otro está basada en el cumplimiento de una condición concreta, hasta que se llega a un nodo hoja final, que indica el valor de la variable buscada. Por ejemplo, en la Figura 6 se puede ver un árbol de decisión muy simple. Permite predecir qué hará una persona en función de si recibe la visita de sus padres, el tiempo que hace y el dinero que tiene. Se puede apreciar que un árbol de decisión es como un conjunto de reglas *if – else*, que se recorren hasta encontrar aquella en la que cuadra el caso particular bajo estudio. Dos de los algoritmos de aprendizaje de árboles de decisión que podemos encontrar en la literatura son:

    - **C4.5** [23], [25], [26] es el algoritmo de árboles de decisión más conocido. Construye el árbol de decisión tomando como criterio el ratio de ganancia para dividir los datos en categorías, siendo esto lo que le diferencia de otros algoritmos.

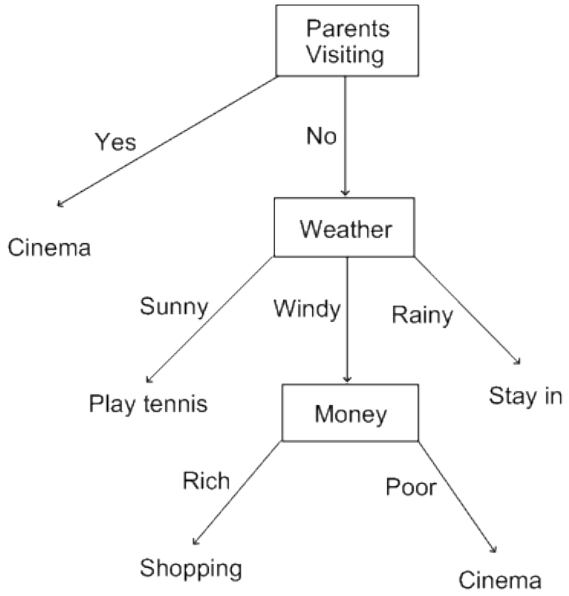


Figura 6. Ejemplo sencillo de árbol de decisión. Fuente: [27]

- ***Random Forest*** [28] se encuadra dentro de una sub-clase de árboles de decisión que no usan un único árbol si no varios árboles sobre lo que se entrena de forma independiente y en paralelo, de tal forma que se consigue reducir la varianza de las decisiones que se tome con ellos. Un esquema puede verse en la Figura 7.

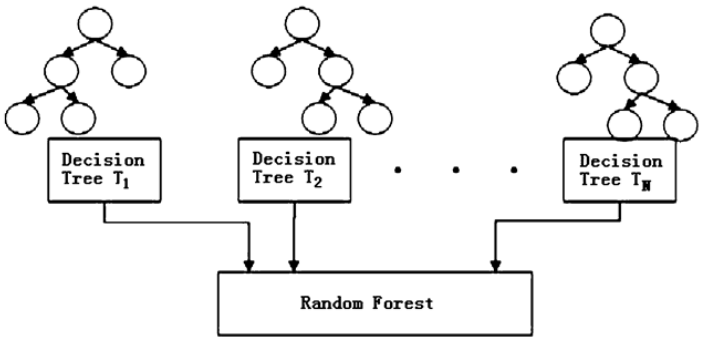


Figura 7. Esquema *Random Forest*. Fuente: [29]

- **Bayes ingenuo (*Naïve Bayes*)**. Este un método muy importante, dada su facilidad de construcción [26]. A pesar de no ser el método por excelencia para aplicaciones concretas, funciona bastante bien en la mayor parte de casos, aun habiendo un algoritmo con mejor rendimiento en cada caso. Es un clasificador de tipo binario (por lo que solo permite clasificar los datos como pertenecientes a una de dos categorías) probabilístico fundamentado en el teorema de *Bayes*. Con los datos de entrenamiento genera un umbral con el que después comparará nuevos datos para asociarlo a la clase caracterizada por superar el umbral o a la caracterizada por no superarlo. Calcula probabilísticamente la función de distribución condicional de cada característica dada una categoría de clasificación, y la aplica al teorema de *Bayes* para calcular la distribución de probabilidad condicionada de la categoría una vez conocida la observación. La probabilidad obtenida se usa en predicción [28].

- **Redes neuronales (*Neural Networks*)** [24], [30]. Una red neuronal consiste en un conjunto de neuronas artificiales conectadas entre sí y que trabajan en conjunto, sin que haya una tarea concreta para cada una. A base de experiencia van aprendiendo y modificando su forma de trabajar para conseguir mejores resultados o resultados adaptados a los cambios que puedan acaecer. Es un método que puede ser tanto supervisado (si se conoce cuál debe ser la salida) como no supervisado (no se conoce la salida), al contrario que el resto de métodos de clasificación que son siempre supervisados.

- ***Support Vector Machine (SVM)*** [31]. Consiste en un conjunto de modelos de aprendizaje supervisado que permiten analizar y reconocer patrones en los datos. Dado un conjunto de datos de entrenamiento, cada uno etiquetado como perteneciente a una de dos categorías, el algoritmo *SVM* construye un nuevo modelo con el que etiquetar nuevos datos, como cualquier otro mecanismo de clasificación, pero sin usar conceptos probabilísticos.

- **K vecinos más cercanos (*K – Nearest Neighbors, KNN*)**. Permite clasificar los datos en una de las posibles categorías generadas con los datos de entrenamiento. Una vez generadas las categorías, cuando llega un nuevo dato, este se clasifica teniendo en cuenta la clasificación de los K puntos de datos más cercanos [24], [26]. Podemos analizar el ejemplo de la Figura 8, donde tratamos de clasificar un nuevo dato ? considerando uno, dos y tres vecinos, respectivamente. En primer lugar, considerando un solo vecino, se decidiría clasificarlo en A, ya que está es la categoría predominante; considerando dos vecinos, no se podría decidir, porque hay igual de vecinos en cada categoría posible; y con tres vecinos se volvería a decidir A por ser la opción predominante. Este tipo de algoritmos se usan sobre todo en procesado de imágenes, donde cada punto de datos se interpreta como cada píxel de la imagen y como vecinos, los píxeles adyacentes. En el caso de usar este método para clasificar datos, se toman como puntos de datos cada una de las entradas del conjunto de datos y se buscan aquellos datos con características o valores similares.

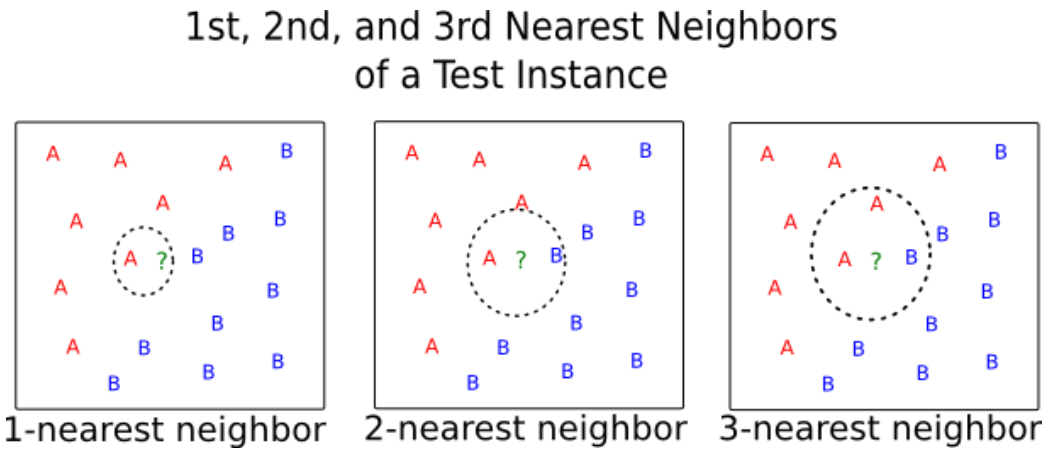


Figura 8. Ejemplo sencillo del algoritmo *K – Nearest Neighbors*. Fuente: [32]

Algoritmos de *clustering* o agrupación. Estos algoritmos se usan para formar grupos de datos similares en base a una determinada característica común [16]. Son algoritmos no supervisados ya que, directamente, con los datos a analizar, en un mismo análisis generan los grupos y clasifican los datos en ellos. Por ejemplo, motores de búsqueda como *Google* o *Yahoo!* usan técnicas de *clustering* para agrupar datos con características similares; también los usan los "grupos de noticias" (*newsgroups*) para agrupar artículos basados en temas relacionados, como en el esquema de la Figura 9, cuando se recibe un nuevo artículo, en este caso un tutorial, el algoritmo de *clustering* trata de decidir dónde debe agruparlo.

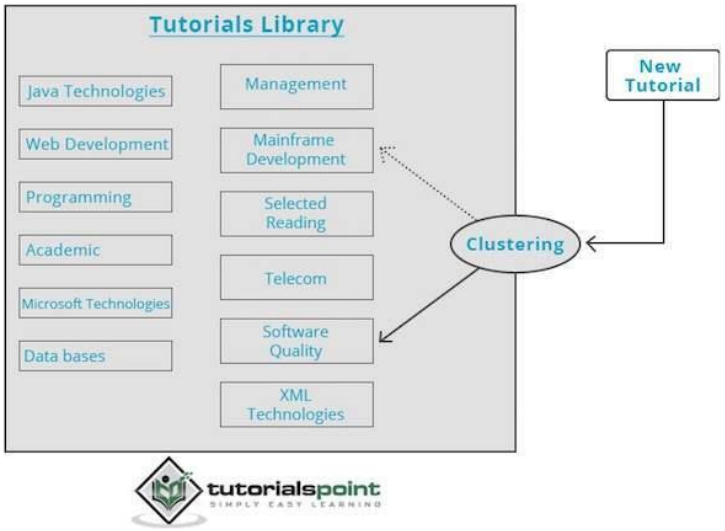


Figura 9. Ejemplo de aplicación de *clustering*. Fuente: [16]

Para implementar un algoritmo de *clustering* se siguen tres pasos: seleccionar el algoritmo en base al cual se agruparán los elementos, definir las reglas de similitud que deben guardar los elementos de un mismo grupo y la condición de *stop* con la que se define el punto donde no se requiere agrupar más.

- **K – Medias (*K – Means*)**. Este es el algoritmo de agrupamiento más simple y popular [15]. Se basa en agrupar los datos de entrada en uno de k *clusters* o grupos en base a sus atributos. En concreto, permite dividir n datos en k grupos, donde cada dato pertenecerá al *cluster* (solo uno) cuyo valor medio se aproxime más al del dato. Para ello [33], en primer lugar, escoge k puntos de datos aleatorios como centro de los correspondientes *clusters*. Después, el resto de datos se asignan al *cluster* cuyo punto central está más cerca del dato. Tras esto, cada punto central del *cluster* se sustituye por la media de los puntos de datos que ya se han asignado a él. Esto puede provocar que algún punto que ya había sido asignado, sea reasignado a otro *cluster*, porque sea otro el más cercano. El proceso sigue hasta que no existe ninguna reasignación. Por ejemplo, en la Figura 10 podemos ver un ejemplo. La imagen de la izquierda representa los puntos de datos a clasificar. El resultado del algoritmo se muestra en la imagen derecha. Con estos datos se han generado tres *clusters* (azul, verde y rojo), cada uno de los cuales se compone de los puntos de datos que tienen valores similares entorno a un mismo valor medio.

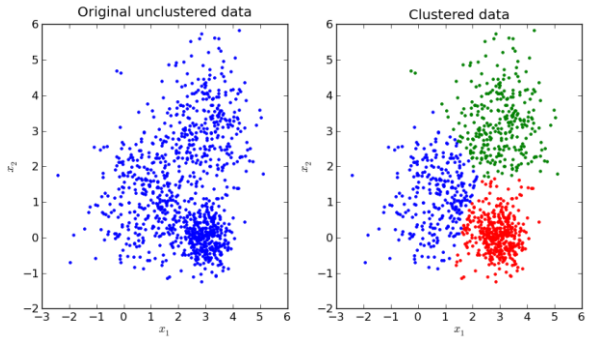


Figura 10. Ejemplo sencillo del algoritmo *K – Means*. Fuente: [34]

- ***Expectation – Maximization, EM***. Este algoritmo [33] busca los *clusters* en los que organizar los datos por medio de una mezcla de funciones *Gaussianas* que modelan un conjunto concreto de datos. Cada una de estas funciones tienen una media y una covarianza. La probabilidad a priori de cada una de estas *Gaussianas* es la fracción de puntos en el *cluster* definidos por esta *Gaussiana*. Estos parámetros pueden ser inicializados seleccionando aleatoriamente la media de las *Gaussianas* o tomando la salida del algoritmo *K – Means* para seleccionar los centros iniciales. Se sigue un proceso iterativo en que se actualiza la media y la covarianza de la función *Gaussiana* en base a los puntos que se van asignando al *cluster* que modela, hasta conseguir una solución óptima. En la Figura 11 se puede ver como ante un mismo conjunto de datos, los algoritmos *K – Means* y *EM* pueden dar resultados distintos.

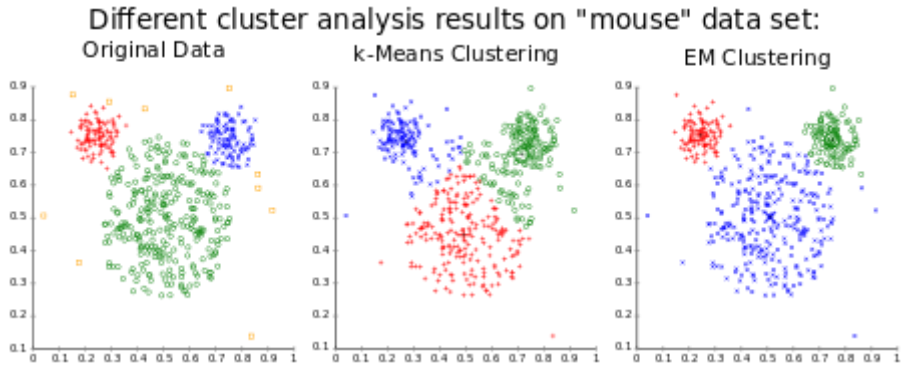


Figura 11. Comparativa entre algoritmos de *clustering K – Means* y *EM*. Fuente: [35]

- ***Clustering* Jerárquico (*Hierarchical Clustering*)**. En este tipo de agrupamiento los datos no se dividen y se asignan en el *cluster* correspondiente en una única etapa, si no que se dividen recursivamente los datos, bien en el sentido de arriba abajo (*top – down*) o de abajo arriba (*bottom – up*) [36]. Es decir, se partiría de un *cluster* que contiene todos los datos y sucesivamente se iría dividiendo ese *cluster* para al final agrupar los datos en n *clusters*, cada uno de ellos conteniendo un objeto. Para poder implementar los dos tipos de sentidos en la recursión existen métodos aglomerativos y métodos divisivos. Los métodos aglomerativos (*bottom – up*) parten de considerar que cada dato conforma un *cluster* para después ir uniendo distintos datos/*clusters* para generar la estructura final. Los métodos divisivos (*top – down*) parten de un único *cluster* donde se agrupan todos los datos para después ir dividiéndolo en sub – *clusters* que sucesivamente siguen sub – dividiéndose hasta conseguir la estructura final. A esta estructura final se le denomina dendrograma y representa los *clusters* finales donde se agrupan los objetos en niveles de similitud. Un ejemplo puede verse en la Figura 12. En ella podemos ver cómo se han ido agrupando cinco datos: *p*, *q*, *r*, *s* y *t*. Si seguimos el sentido de agrupación divisivo, partimos de un único *cluster* que agrupa los cinco datos. Este *cluster* se divide en otros dos sub – *cluster*, el primero de ellos agrupa los datos *p* y *q*; y el otro los datos *r*, *s* y *t*. Este último a su vez se divide en otros dos *clusters*, uno que agrupa el dato *r* y otro que agrupa los datos *s* y *t*. Los *clusters* que agrupan dos datos, a su vez se dividen en *clusters* que agrupan individualmente cada uno de los datos. En el sentido aglomerativo, se partiría de un *cluster* por dato para llegar a un *cluster* que agrupe todos.

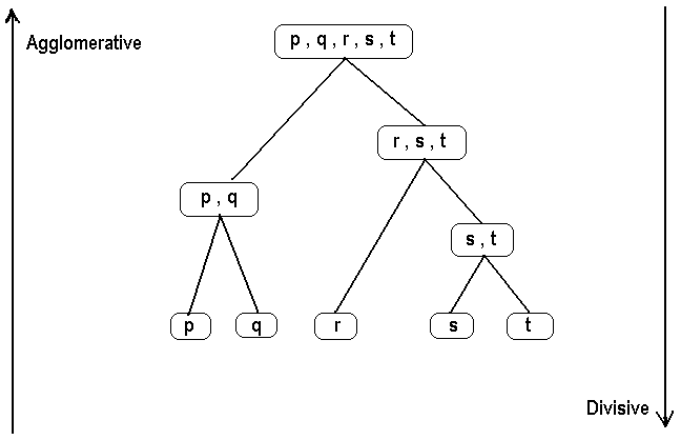


Figura 12. Ejemplo sencillo de *clustering* jerárquico. Fuente: [37]

Algoritmos de regresión. La regresión es un proceso estadístico que permite estimar relaciones entre variables. Es bastante común encontrar conjuntamente definidos los algoritmos de regresión y clasificación. La principal diferencia entre estos dos conceptos está en que la salida de un algoritmo de regresión toma valores continuos, mientras que en la clasificación solo puede tomar un conjunto concreto de valores: las etiquetas de las clases correspondientes. Como algoritmos de regresión nos encontramos prácticamente los mismos que para clasificación, principalmente: árboles de decisión, *Bayes*, *K – Nearest Neighbors*, redes neuronales y *SVM*. También se pueden destacar otros como los que a continuación se citan.

- **Regresión logística**. Es un tipo de análisis usado para predecir el resultado de una variable categórica, es decir que solo puede tomar un valor de entre un conjunto finito de posibilidades.

- **Regresión por mínimos cuadrados**. Consiste en el uso del algoritmo de mínimos cuadrados para resolver problemas estadísticos y problemas de regresión.

El **Análisis de Secuencias** [12], [15], [38] puede considerarse una extensión más restrictiva de los algoritmos para la obtención de reglas de asociación donde no solamente importa los elementos que conforman las reglas si no también el orden en el que ocurrieron. Con este conjunto de algoritmos se pretende descubrir si la presencia de un conjunto de elementos es seguido por otro conjunto de elementos en orden cronológico a lo largo de un conjunto de sesiones independientes. Debido a su similitud con las reglas de asociación, también se puede realizar análisis de secuencias con el algoritmo *Apriori*. Existe otro algoritmo "propio" del análisis de secuencias denominado ***PrefixSpan*** (*Prefix – projected Sequential Pattern Mining*) [39]. Este algoritmo, en vez de buscar ocurrencias de secuencias o subsecuencias frecuentes, se basa en prefijos, ya que cualquier secuencia frecuente se puede buscar por medio de un prefijo más o menos largo, según la situación, de forma que se consigue mejorar la eficiencia.

En este apartado se han analizado algunas de las categorías de algoritmos y los algoritmos que podemos encontrar en ellas, principalmente las más usuales, conocidas o las opciones que podríamos usar en nuestro entorno *BigData* (ver sección 2.2.3.2.5), pero existen otras muchas opciones.

2.1.2. *Educational Data Mining*

Como parte de la Minería de Datos, surge la **Minería de Datos Educativa** (*Educational Data Mining*, EDM), una disciplina que se encarga de aplicar los conceptos de *Data Mining* al contexto educativo. Por tanto, se centrará en analizar la gran cantidad de datos que estudiantes y profesores generan con el objetivo de entender mejor su comportamiento y la forma de aprendizaje [40]. Principalmente, estos datos se extraen de una plataforma de educación tecnológica, como un sistema *LMS* (*Moodle*, *Blackboard*) o sistemas de tutoría inteligente (*Intelligent Tutoring System*, ITS).

Existen diversas forma de clasificar los trabajos típicos realizados en EDM, por ejemplo, [41] considera que en el contexto educativo, las técnicas *Data Mining* se suelen usar, principalmente, para:

- Generar sistemas de predicción mediante algoritmos de clasificación o regresión.
- Agrupamiento o *clustering*.
- Buscar relaciones mediante el uso de reglas de asociación, búsqueda de patrones secuenciales o de correlaciones.

Romero, Ventura y Hervás [12] consideran que las principales aplicaciones de EDM son, entre otras:

- Sistemas de recomendación.
- Sistemas de personalización.
- Sistemas de detección de irregularidades.
- Clasificaciones de estudiantes y contenidos.
- Descubrimiento de relaciones entre actividades.

Para lo que fundamentalmente se usan algoritmos de:

- Reglas de asociación.
- Clasificación.
- *Clustering*.

El peso de los diferentes algoritmos ha ido modificándose a lo largo del tiempo a medida que surgían nuevas técnicas.

Romero y Ventura [42] realizaron en el año 2010 un estudio de todas las investigaciones que se habían hecho en este campo, diferenciando hasta 11 categorías de trabajo distintas en las clasificaron cerca de 300 trabajos de otros autores, poniendo de manifiesto que este es un área de investigación muy activo y en constante evolución, ya que surgen nuevas formas de hacer las cosas o la necesidad de nuevos objetivos. Estos 11 objetivos de trabajo se listan a continuación:

- **Análisis y visualización de datos**, con el objetivo de remarcar los puntos importantes de la información para que, después, el usuario que los consulta pueda tomar decisiones o sacar conclusiones. Básicamente puede considerarse como un **análisis estadístico**

de los datos que extraemos del contexto educativo correspondiente, sin aplicar técnicas de DM utilizando directamente los datos extraídos de la fuente.

- **Proporcionar realimentación** a los profesores o administradores con el objetivo de poder tomar decisiones, pero a través de unos datos a los que se ha aplicado algún mecanismo de ML y que, por lo tanto, tienen un mayor sentido lógico. Cómo principal mecanismo ML se suelen usar reglas de asociación.

- Realizar **recomendaciones** a los estudiantes proponiéndoles actividades personalizadas, visitar enlaces, realizar tareas o problemas. Otro objetivo a añadir a esta lista sería la capacidad de ofrecer contenidos adaptados a cada estudiante en particular. Para todo ello, principalmente se usan reglas de asociación, y técnicas de *clustering* y análisis de secuencias.

- **Predicción del rendimiento** de los estudiantes medido mediante métricas como su conocimiento, resultados o nota final. Estas métricas se pueden evaluar de forma que tomen un valor continuo, usando técnicas de regresión, o de forma discreta, mediante técnicas de clasificación. Es decir, si se usan técnicas de regresión podremos medir la nota del estudiante con cualquier valor numérico entre unos límites superior e inferior; y usando técnicas de clasificación se categorizará la nota como perteneciente a una clase (de entre un conjunto finito de clases), por ejemplo, clase A que puede indicar que la nota está entre 9 y 10. Como técnicas de clasificación y regresión se suelen usar redes neuronales, redes bayesianas o reglas de asociación.

- **Modelado de los estudiantes**, cuyo objetivo es generar modelos con los que podamos caracterizar el comportamiento, modo de aprendizaje o conocimiento de los estudiantes. Para conseguir este objetivo, se suelen usar técnicas de clasificación como redes bayesianas, reglas de asociación o análisis de secuencias.

- **Detección de comportamiento indeseable** en los estudiantes para conocer aquellos estudiantes que tienen problemas de aprendizaje, hacen un mal uso de la plataforma educativa, no están motivados con las asignaturas, etc. Para llevar a cabo esta **detección de *outliers*** se usan técnicas de clasificación y *clustering*.

- **Agrupación de estudiantes** para crear grupos en los que se clasifiquen los estudiantes de acuerdo a multitud de características como rendimiento, nota final, uso de la plataforma, capacidades, etc. Estas clasificaciones pueden ser útiles, por ejemplo, para que un profesor pueda intuir cómo trabaja cada uno de sus alumnos y decidir cómo realizar los grupos de trabajo. Para agrupar estudiantes se usan técnicas de clasificación, si se quiere tomar un modelo de datos de partida en función del cual clasificar los estudiantes, o técnicas de *clustering*, si será el propio algoritmo el que determine cómo categorizar a los estudiantes. Normalmente las técnicas de *clustering* suelen generar grupos con un número similar de elementos, siempre que los datos lo permitan.

- **Análisis de redes sociales**. Por red social se considera a un conjunto de personas que están conectadas por medio de una relación social como amistad o una relación cooperativa. El análisis de redes sociales pretende estudiar las relaciones entre estos individuos usando, principalmente, técnicas de filtrado colaborativo.

- **Desarrollo de mapas conceptuales**. Un mapa conceptual se define como un gráfico que muestra las relaciones entre conceptos y expresa la estructura jerárquica del conocimiento. Se usan reglas de asociación y minería de texto como principales herramientas.

- **Construcción de cursos**. Se están desarrollando o utilizando técnicas de DM para facilitar al profesor (o quien corresponda) la construcción del curso, es decir, la selección de los contenidos que se deben ofrecer a los alumnos.

- **Planificación y programación**. Existen trabajos cuyo objetivo es mejorar el proceso educativo tradicional utilizando técnicas de DM (principalmente reglas de asociación) para permitir planificar los cursos futuros, ayudar a la planificación de trabajo del estudiante, planificar la asignación de recursos, ayudar en la admisión y el asesoramiento a la hora de desarrollar un plan de estudios, entre otros.

En la Figura 13 se muestra un gráfico, adaptado de [42], en el que se pueden ver los estudios realizados (medido en base al número de artículos publicados) sobre cada uno de estos temas de trabajo, hasta el año 2009, y en función del estudio realizado por [42]. En este gráfico podemos ver que los trabajos más comunes están relacionados con el análisis estadístico de los datos, ofrecer realimentación a los profesores, sistemas de recomendación y sistemas de predicción.

A nuestro juicio y en relación al trabajo que vamos a realizar, teniendo en cuenta cuáles son sus objetivos, nos parecen interesantes las siguientes categorías:

- Análisis estadístico.
- Sistemas de recomendación.
- Sistemas de predicción.
- Agrupación de estudiantes.
- Clasificación de contenidos.
- Detección de anomalías.

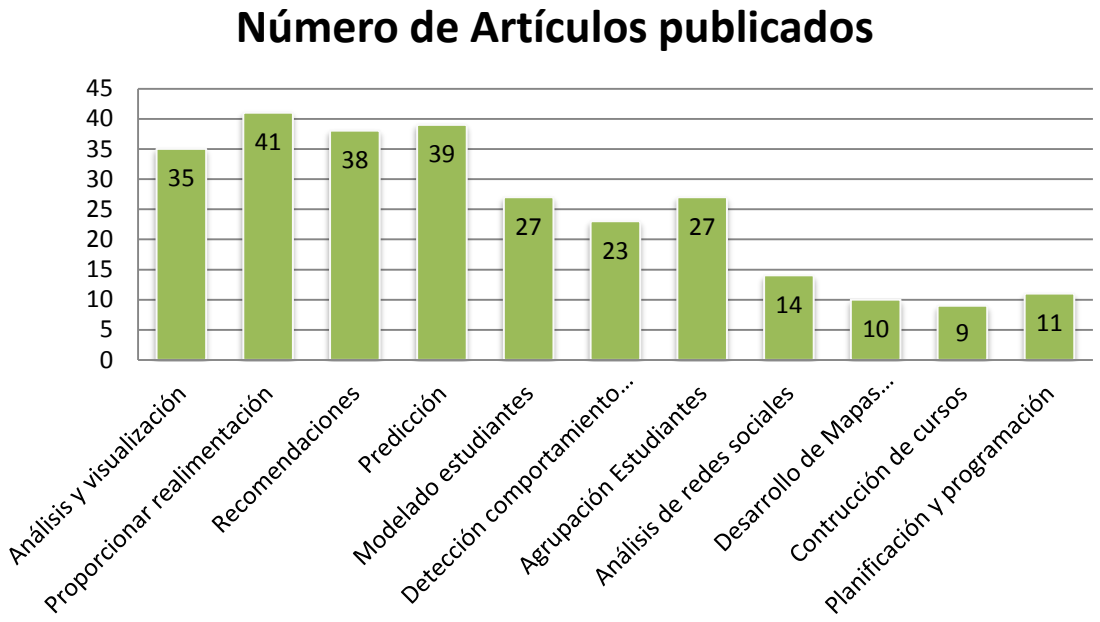


Figura 13. Número de artículos publicados hasta el año 2009 en el contexto de EDM agrupados por categoría del trabajo realizado. Fuente: Adaptado de [42]

La categoría "Proporcionar Realimentación" englobaría cualquiera de las anteriores, ya que todos nuestros objetivos van a estar centrados en proporcionar datos a los profesores que puedan ayudarles a mejorar el desarrollo de sus asignaturas, e incluso proporcionar ciertos resultados personalizados a los alumnos. Por otro lado, el modelado y clasificación de estudiantes, podrían considerarse muy similares, y por ello, lo vamos a englobar dentro de nuestra categoría de agrupación de estudiantes. Las últimas cuatro categorías (Análisis de Redes Sociales, Desarrollo de Mapas Conceptuales, Planificación y Construcción del Curso) no son de nuestro interés en estos momentos.

Teniendo en cuenta estas cuestiones, analizamos distintos artículos cuyo contenido nos pareció interesante por aportar alguna idea que podía ser de utilidad. El resto de la sección se centrará en describir los estudios analizados.

Casey, Gibson y Paris [10] utilizan los registros de actividad de los alumnos (389 alumnos en 13 cursos) en *Moodle* para realizar un análisis estadístico. Para ello, realizan los siguientes cálculos, y buscan la correlación con la nota final del estudiante:

- Número total de visitas a páginas realizadas por cada estudiante a lo largo de un semestre.
- Número total de visitas a páginas únicas realizadas por cada estudiante.
- Número de visitas diarias a la página principal de cada curso realizadas por cada estudiante.
- Número y porcentaje de visitas al total de recursos que cada estudiante realiza, en cada curso, desde dentro y fuera del campus.
- Número de recursos leídos al menos una vez por el estudiante por curso.
- Relación entre el número de lecturas de un recurso y la calificación final.
- Análisis de la actividad de los usuarios en *Moodle* según el día de la semana medido en función del porcentaje de accesos por día.
- Cálculo del tiempo de primera visita a un recurso (desde su publicación) en relación a la media de todos los estudiantes.

Con este análisis, los autores consiguen encontrar qué actividades contribuyen más y mejor a la nota final de los estudiantes, así como, cómo de diferente es la actividad en los distintos cursos y cómo de diferente afecta la misma actividad en distintos cursos, es decir, existen cursos en los que existe una mayor correlación entre el número de visitas por estudiante y la nota final, lo cual puede significar que para los estudiantes, las visitas son más provechosas a un curso que a otro, porque puedan tener mejores recursos, el profesor actualice la información más frecuentemente, etc; por otro lado, en la mayoría de cursos, más del 50% de las visitas se realizan desde dentro del campus, pero existen un par de casos en que es al revés, lo cual puede implicar que en esas asignaturas los alumnos necesitan mayor trabajo en casa o la asignatura es plenamente teórica y no pueden acceder durante las horas de clase por estar en un aula sin ordenadores.

Romero, Ventura y García [15], proponen utilizar los registros de actividad de *Moodle* para aplicar algoritmos de *Machine Learning*, considerando datos de 438 estudiantes en 7 cursos. A partir de estos registros, y por estudiante, se obtienen los siguientes datos:

- Identificador del curso (`Course`).

- Número de tareas realizadas (`n_assignments`).
- Número de cuestionarios realizados (`n_quiz`).
- Número de cuestionarios aprobados (`n_quiz_a`).
- Número de cuestionarios suspendidos (`n_quiz_s`).
- Número de mensajes enviados al chat (`n_messages`).
- Número de mensajes enviados al profesor (`n_messages_ap`).
- Número de mensajes enviados al foro (`n_posts`).
- Número de mensajes leídos del foro (`n_read`).
- Tiempo total dedicado a tareas (`total_time_assignment`).
- Tiempo total dedicado en cuestionarios (`total_time_quiz`).
- Tiempo total dedicado en foros (`total_time_forum`).
- Nota final del estudiante (`Mark`).

Este primer paso, pone de manifiesto el hecho de que el análisis estadístico va a ser siempre necesario como paso previo para obtener datos o métricas de interés para poder aplicar a procedimientos de Minería de Datos. Por ello, puede ser interesante, no solamente generar métricas como las anteriores para implementar un algoritmo de *Machine Learning*, sino también utilizar estos resultados "intermedios" como finales y mostrárselos al usuario junto con los resultados de la Minería de Datos.

Otro punto importante a considerar es el pre – procesado de los datos, ya comentado en la sección 2.1. Por lo general, este pre – procesado va a consistir en la discretización de los valores numéricos. Con los datos considerados en este artículo, esto se realiza en dos pasos:

- La nota final del estudiante se categorizará como *FAIL* (nota < 5), *PASS* (5 < nota < 7), *GOOD* (7 < nota < 9) y *EXCELLENT* (nota > 9).
- El resto de atributos se categorizarán como *LOW*, *MEDIUM* o *HIGH* dividiendo su rango de extensión en tres intervalos de igual longitud.

Con los datos ya discretizados, aplica tres técnicas de ML:

- *Clustering* (*K – Means*) para agrupar alumnos de un curso en función de las actividades realizadas en *Moodle* y en relación con su nota final. Con los datos del caso de estudio obtiene que sus alumnos quedan agrupados en tres *clusters*:
    - Estudiantes muy activos.
    - Estudiantes activos.
    - Estudiantes no activos.
- Clasificación (C4.5) para implementar un sistema de predicción de la nota final del estudiante. Al implementarlo mediante un árbol de decisión obtienen un conjunto de re-

glas *IF-THEN-ELSE* (Figura 14) con las que ir recorriendo el árbol y decidir la nota final del estudiante (*FAIL*, *PASS*, *GOOD* o *EXCELLENT*) en función al resto de datos.

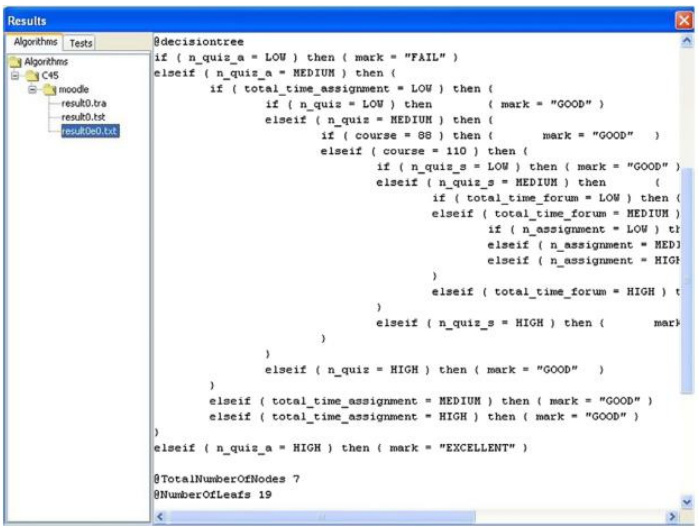


Figura 14. Ejecución Algoritmo C4.5. Fuente: [15]

- Reglas de asociación (*Apriori*) para buscar relaciones entre los distintos parámetros, obteniéndose resultados como:
    - Si un estudiante no envía mensajes al foro, tampoco los leerá.

$$n\_post = LOW => n\_read = LOW$$

    - Si el número de mensajes leídos y enviados a los foros es bajo, la nota final será suspenso.

$$n\_post = LOW \ \& \ n\_read = LOW => mark = FAIL$$

De forma análoga, Romero, González, Ventura, del Jesús y Herrera [43], utilizan casi los mismos datos (293 estudiantes y 5 cursos):

- Identificador del curso (`Course`).
- Número de tareas realizadas (`n_assignments`).
- Número de tareas aprobadas (`n_assignments_a`).
- Número de tareas suspendidas (`n_assignmemts_s`).
- Número de cuestionarios realizados (`n_quiz`).
- Número de cuestionarios aprobados (`n_quiz_a`).
- Número de cuestionarios suspendidos (`n_quiz_s`).
- Número de mensajes enviados al chat (`n_messages`).
- Número de mensajes enviados al profesor (`n_messages_ap`).
- Número de mensajes enviados al foro (`n_posts`).

- Número de mensajes leídos del foro (`n_read`).
- Nota final del estudiante (`Mark`).

Para buscar relaciones entre cómo los estudiantes usan *Moodle* y las calificaciones finales que obtienen, usando reglas de asociación y comparando distintos algoritmos. Así, obtienen reglas del tipo:

- Si el número de cuestionarios aprobados es muy bajo, entonces la nota final será FAIL.

$$IF\ n\_quiz\_a\ =\ VERY\ LOW\ THEN\ mark\ =\ FAIL$$

- Si el número de cuestionarios aprobados es muy alto, entonces la nota será EXCELLENT.

$$IF\ n\_quiz\_a\ =\ VERY\ HIGH\ THEN\ mark\ =\ EXCELLENT$$

- Si estamos en un curso concreto, el número de mensajes publicados en foros es alto o muy alto y el número de cuestionarios aprobados es medio, alto o muy alto, entonces la nota final será GOOD.

$$IF\ course\ =\ C110\ o\ C88\ AND\ n\_posts\ =\ HIGH\ OR\ VERY\ HIGH\ AND\ n\_quiz\_a\ =\ MEDIUM\ OR\ HIGH\ OR\ VERY\ HIGH\ THEN\ mark\ =\ GOOD$$

López, Luna, Romero y Ventura [44] proponen una forma de clasificación mediante *clustering* para decidir la calificación final que obtendrán los alumnos en base a su participación en los foros de la asignatura. Para ello, analizan los datos de 114 estudiantes, los cuales han publicado 1014 mensajes, en 81 hilos con 922 respuestas en total. Toma como datos:

- Número de mensajes enviados por los estudiantes (`nMessages`).
- Número de hilos creados por un estudiante (`nThreads`).
- Número de réplicas enviadas por estudiante (`nReplies`).
- Número de palabras escritas por estudiante (`nWords`).
- Número de sentencias frases por estudiante (`nSentences`).
- Número de mensajes leídos en el foro (`nReads`).
- Tiempo total en horas dedicado al foro (`tTime`).
- Media de la calificación de los mensajes (`aEvaluation`).
- Grado de centralidad del estudiante (`dCentrality`), es decir, el nivel de relación del estudiante con otros estudiantes.
- Grado de prestigio del estudiante (`dPrestige`), una medida cómo se involucra socialmente el estudiante, con más importancia que el parámetro de centralidad.
- Nota final del estudiante (`fMark`).

Por un lado, podemos obtener ideas de cómo a partir del uso de los foros y las métricas anteriores se puede implementar un mecanismo de predicción de la nota final, pero lo interesante de este artículo está en las conclusiones, ya que nos va a permitir centrar procedimientos típicos. Normalmente los sistemas de predicción se implementan mediante algoritmos de cla-

sificación. En este artículo se analizan distintos algoritmos de *clustering* y se concluye que la predicción con ellos es bastante pobre, comparada con la que ofrecen los algoritmos de clasificación, también probados en el estudio. Compara ambos conjuntos de algoritmos en base a su precisión entre la nota predicha y la nota real obtenida por el estudiante. Todos los algoritmos de clasificación presentan una precisión superior al 80 %, mientras que los algoritmos de *clustering* tienen una precisión entre el 50 y 80 % en el mejor de los casos.

Un par de nuevos artículos de Romero y otros [23], [45] no ofrecen nada todavía no conocido, ya que presentan un resumen de las técnicas de clasificación que se pueden usar en *Data Mining* para predecir la nota final de un estudiante en base a su uso de la plataforma *Moodle*. Realmente, en este artículo, presentan una herramienta que han generado y que se puede integrar en *Moodle* y que permite que profesores que no conozcan ni sepan usar la Minería de Datos, puedan aplicar algoritmos de ML de análisis estadístico, *clustering*, reglas de asociación o clasificación, de una forma muy sencilla. Para ello, siguen los pasos del artículo [15], previamente comentado, y generan una herramienta con la que el profesor pueda, en primer lugar escoger una asignatura, indicar las notas de sus estudiantes, seleccionar el tipo de procesamiento que quieren realizar y con qué datos, para obtener el resultado del análisis correspondiente.

Lo realmente interesante de este artículo está en una serie de cuestiones a considerar para conseguir resultados óptimos. Una parte fundamental de la minería de datos es el pre – procesamiento de los datos a utilizar. Un buen pre – procesado nos puede ayudar a conseguir los mejores resultados. En este artículo se analizó la importancia de esta etapa en la clasificación de los estudiantes, comparando los resultados de tres estudios:

- Datos originales.
- Datos discretizados, como se ha comentado anteriormente.
- Datos balanceados. Se dice que los datos no están balanceados cuando con ellos se generan clases con muy pocos elementos clasificados en ellas. Esto es algo muy común si clasificamos alumnos por sus notas, será bastante frecuente tener alumnos con notas "medias" pero muy pocos alcanzarán la categoría de excelencia. Balancear los datos tendrá por objetivo que con ellos se generen clases que, más o menos, estén compuestas por el mismo número de instancias.

Para evaluar con qué datos se obtienen mejores resultados, utiliza como métrica el porcentaje de datos que se han clasificado correctamente. El resultado concreto depende del algoritmos de clasificación usado (testea hasta 21 algoritmos de clasificación distintos para los tres conjuntos de datos), pero por lo general, utilizar el último conjunto de datos da los peores resultados, lo cual es obvio ya que se están falseando los datos para que las categorías finales sean parecidas en número de elementos. Entre utilizar el primero o segundo conjunto de datos, depende de cada algoritmo, pero las diferencias no son muy grandes. Puede verse la tabla de resultados en la Figura 15.

Table 3 Classification Results With Numerical, Categorical, and Rebalanced Data (Global Percentage of Correctly Classified/Geometric Mean)

Method	Algorithm	Numerical data	Categorical data	Rebalanced data
Statistical Classifier	ADLinear	59.82/0.00	61.66/0.00	59.82/0.00
Statistical Classifier	PolQuadraticLMS	64.30/15.92	63.94/18.23	54.33/26.23
Statistical Classifier	Kernel	54.79/0.00	56.44/0.00	54.34/0.00
Statistical Classifier	KNN	59.38/10.15	59.82/7.72	54.34/10.21
Decision Tree	C45	64.61/41.42	65.29/18.10	53.39/9.37
Decision Tree	CART	65.77/39.25	65.86/24.54	47.51/34.65
Rule Induction	AprioriC	60.04/0.00	59.82/0.00	61.64/0.00
Rule Induction	CN2	64.17/0.00	63.47/3.52	50.24/15.16
Rule Induction	Corcoran	62.55/0.00	64.17/0.00	61.42/0.00
Rule Induction	XCS	62.80/0.00	62.57/0.00	60.04/23.23
Rule Induction	GGP	65.51/1.35	64.97/1.16	52.91/12.63
Rule Induction	SIA	57.98/0.00	60.53/0.00	56.61/15.41
Fuzzy Rule Learning	MaxLogitBoost	64.85/0.00	61.65/0.00	62.11/8.83
Fuzzy Rule Learning	SAP	63.46/0.00	64.40/0.00	47.23/3.20
Fuzzy Rule Learning	GAP	65.99/0.00	63.02/0.00	52.95/26.65
Fuzzy Rule Learning	GP	63.69/0.00	63.03/0.00	53.19/11.97
Fuzzy Rule Learning	Chi	57.78/10.26	60.24/0.00	41.11/14.32
Neural Networks	NNEP	65.95/0.00	63.49/0.00	54.55/12.70
Neural Networks	RBFN	55.96/3.23	54.60/0.00	37.16/4.00
Neural Networks	GANN	60.28/0.00	61.90/4.82	53.43/17.33
Neural Networks	MLPerceptron	63.91/9.65	61.88/4.59	53.21/17.16

Figura 15. Comparativa de resultados según pre – procesado de datos. Fuente: [23]

Otro punto importante, aparte de cómo manipular previamente los datos, está en qué datos seleccionar, es decir, no escoger toda la información disponible sino solo aquella que nos pueda ser de interés al aplicar el algoritmo, ya que, a lo mejor, la información a mayores que se proporciona al algoritmo, puede hacer que el resultado no sea el mejor. En este sentido se pueden hacer dos cosas:

- Eliminar *outliers*, aquellos datos que son muy diferentes a los demás y que pueden confundir al algoritmo. Por ejemplo, eliminar estudiantes que no han completado todas las actividades.

- Utilizar todos los datos (todos los estudiantes) pero escogiendo solo los atributos de su actividad más relevantes.

Table 2 Classification Results With All Available Data and Filtered Data (Global Percentage of Correctly Classified)

Method	Algorithm	All available data	Filtered data by row	Filtered data by column
Statistical Classifier	ADLinear	59.82	28.13	63.49
Statistical Classifier	PolQuadraticLMS	64.30	50.93	63.03
Statistical Classifier	Kernel	54.79	33.18	58.00
Statistical Classifier	KNN	59.38	49.12	60.51
Decision Tree	C45	64.61	45.16	63.01
Decision Tree	CART	65.77	40.71	64.15
Rule Induction	AprioriC	60.04	35.60	59.82
Rule Induction	CN2	64.17	39.28	63.46
Rule Induction	Corcoran	62.55	36.86	58.91
Rule Induction	XCS	62.80	47.30	62.34
Rule Induction	GGP	65.51	63.44	43.86
Rule Induction	SIA	57.98	44.94	61.19
Fuzzy Rule Learning	MaxLogitBoost	64.85	43.68	63.23
Fuzzy Rule Learning	SAP	63.46	38.62	64.16
Fuzzy Rule Learning	GAP	65.99	43.46	64.15
Fuzzy Rule Learning	GP	63.69	36.20	63.48
Fuzzy Rule Learning	Chi	57.78	34.06	58.91
Neural Networks	NNEP	65.95	44.50	63.49
Neural Networks	RBFN	55.96	26.92	54.13
Neural Networks	GANN	60.28	42.03	61.43
Neural Networks	MLPerceptron	63.91	45.93	62.34

Figura 16. Comparativa de resultados según filtrado de datos. Fuente: [23]

Exactamente igual que antes, aplica los 21 algoritmos de clasificación con estos dos nuevos casos y obtiene los resultados de la Figura 16. La eliminación de *outliers* (*filtered data by* row)

no permite mejorar los resultados (es más, los empeorá bastante en la mayoría de los casos), pero sí (en ocasiones) la selección de atributos (*filtered data by column*)

Kazanidis y Karakos [46] describen el uso de técnicas de *Data Mining* para analizar los ficheros de *log* de una plataforma de *e – learning* para poder ver la actividad de los cursos con datos de 1199 estudiantes y 39 cursos. A partir de la información genérica de un *log* de un servidor *web Apache*, los datos son filtrados para eliminar *outliers* y quedarse solo con la información más relevante con la que poder obtener las siguientes métricas:

- Número total de sesiones por curso.
- Número total de páginas por curso visitadas por los usuarios.
- Número total de páginas únicas por curso visitadas por los usuarios (contabiliza cada página del curso solo una vez, independientemente de cuántas veces fue visitada).
- Número total de páginas únicas por curso y por sesión visitadas por los usuarios (UPCS). Esta métrica contabiliza las visitas del mismo usuario a la misma página en la misma sesión como una sola visita.
- Enriquecimiento de cursos: 1 – páginas únicas / páginas totales.
- Homogeneidad de cursos: páginas únicas / total sesiones.
- Calidad: media del enriquecimiento y homogeneidad de cursos.

Con estos datos aplica un algoritmo de clasificación en tres etapas. Primero clasifica los cursos en una de dos categorías en función del enriquecimiento (alto o bajo). El enriquecimiento se puede entender como la riqueza de los recursos publicados en el curso. Después, divide cada una de estas dos categorías en otras dos, en función de su homogeneidad (curso estático o con contenido actualizado frecuentemente). Por último, divide cada una de las cuatro categorías en otras dos, en función del UPCS (muy visitado o no por los estudiantes). Estas tres etapas dan lugar a una clasificación de cursos en una de ocho categorías (Figura 17)

Cluster ID	Enrichment	Homogeneity	UPCS	Course Classification
I	High	Frequently Updated	High	Frequent course content updates visited by users
II	High	Frequently Updated	Low	Frequent course content updates not visited by users
III	High	Static Content	High	Static content visited frequently by users
IV	High	Static Content	Low	Static content visited occasionally by users
V	Low	Frequently Updated	High	Abandoned course open for view
VI	Low	Frequently Updated	Low	Garbage course that needs further justification
VII	Low	Static Content	High	Poor content that still contains useful information for students
VIII	Low	Static Content	Low	Abandoned course

Figura 17. Clasificación de cursos final en [46]

A parte de este estudio, también realiza un agrupamiento de cursos en función de su calidad medida en base a la nota obtenida por sus estudiantes (*clustering K – Means*) y busca reglas de asociación (*Apriori*) entre los atributos que forman parte del estudio.

Delgado, Gibaja, Pegalajar y Pérez [47] utilizan redes neuronales para predecir la nota final del estudiante en base a los registros de acceso a *Moodle* de 240 estudiantes. Al no tener disponibles las redes neuronales en nuestra infraestructura *BigData*, lo que nos interesa de este artículo es conocer qué datos se pueden utilizar de partida y qué resultados se pueden obtener de ellos. Lo primero que indica, y se puede apreciar en cualquiera de los estudios que se enumeren en este apartado, es que no solamente la información de la actividad de los estudiantes es necesaria, también se necesitan otros datos de tipo administrativo, como las notas que obtiene el estudiante en las distintas actividades y la nota final de la asignatura o el número de veces que el estudiante ha estado matriculado en el mismo curso o lo que es lo mismo si está repitiendo o no.

Por si pudieran ser de interés en la realización de nuestro trabajo, anotamos los datos que se pueden extraer de la información de cada estudiante:

- Nombre completo.
- Número de veces que ha estado oficialmente registrado en la asignatura.
- Número de sesiones de examinación (suponemos, número de veces que se ha examinado).
- Nota.
- Número total de accesos de cualquier tipo hechos en *Moodle* en el curso.
- Número total de accesos a *resource view*.
- Porcentaje de accesos *resource view* con respecto al total.
- Número de accesos a *resource view* diferentes.
- Porcentaje de *recource view* diferentes.
- Segmentación del número de accesos en cada mes del año.
- Segmentación por meses del porcentaje de accesos.

Meceron y Yacef [48] comparan distintas reglas de asociación usando los datos de *Moodle*, con datos de un curso y 84 estudiantes. Tratan de encontrar:

- Si existe relación entre los accesoa a distintos recursos extras como exámenes de prueba, soluciones a estos, libros, páginas de interés. Obtienen resultados de la forma: Si un estudiante accede al examen de prueba 1, también visitará la solución al examen de prueba 1.
- Si existe relación entre las calificaciones de cada uno de los distintos ejercicios del examen. Cada estudiante tendrá calificado cada ejercicio como resuelto correctamente, resuelto incorrectamente o no intentado. De esta forma, se obtienen reglas del tipo: si un alumno no ha intentado resolver el ejercicio 2, tampoco ha intentado el 3; lo cual puede servir para analizar si los ejercicios del examen estaban preparados correctamente.

Falakmasir y Habibi [49] analizan los registros de acceso a *Moodle* para realizar un ranking con las actividades de *Moodle* (clases virtuales, vista de ficheros, lecturas de mensajes en el foro, etc.) que más han beneficiado a los alumnos con respecto a su nota final, con datos de 824 alumnos en 11 cursos. A partir de los datos de *Moodle* genera los siguientes datos:

- Nombre de estudiante.
- Curso.
- Número de recursos visitados.
- Número de participaciones en *Virtual Classroom.*
- Número de *Archive Views.*
- Número de *Forum Reads.*
- Número de *Forum Post.*
- Número de *Discussion Read.*
- Número de *Discussion Response.*
- Número de *Assignments Views.*
- Número de *Assignments Answer Uploads.*
- Nota final, en forma discreta en cuatro categorías.

Como método *Data Mining* aplica *Feature Selection*, que intenta seleccionar las características más relevantes de acuerdo a un concepto. Con ello, obtiene un ranking de actividades de más a menos en función de su importancia en la nota del estudiante. Por ejemplo, obtiene los resultados de la Figura 18. Obviamente, aplicando distintos métodos de selección obtendrá una clasificación distinta, como se aprecia en la Figura 19.

Table 2: The results of ranking activities based on gain ratio metric

Attribute	Gain Ratio
Virtual Classroom	0.0839
Archive View	0.0694
Forum Read	0.052
Assignment View	0.0517
Assignment Upload	0.0497
Discussion Read	0.0364
Resource View	0.0324
Forum Post	0
Discussion Post	0

Figura 18. Ranking de actividades en [49] I

Table 3: The The results of ranking activities based on other methods

Attribute	χ^2	Info-Gain	Symmetric Uncertainty	One-R	Relief-F	SVM
Virtual Classroom	1	1	1	2	2	2
Archive View	3	3	2	1	3	7
Forum Read	2	2	3	4	7	1
Assignment View	7	7	6	5	4	6
Assignment Upload	4	4	4	3	5	5
Discussion Read	5	5	5	7	6	4
Resource View	6	6	7	8	1	9
Forum Post	8	8	8	9	9	8
Discussion Post	9	9	9	6	8	3

Figura 19. Ranking de actividades en [49] II

Además, con estos resultados, utilizando C4.5 como técnica de clasificación, generan un árbol de decisión con el que poder predecir la nota final de los estudiantes, como se muestra en la Figura 20, donde *Mode* se refiere a la categoría de la nota.

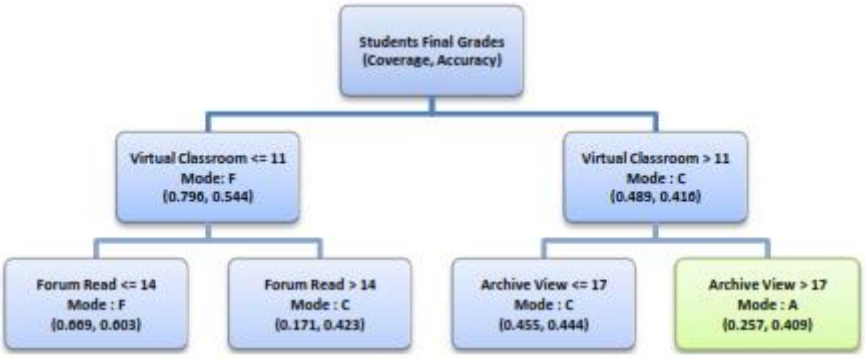


Figura 20. Árbol de decisión para predecir nota en [49]

Zaïne [50] implementa un sistema de recomendación que propone al alumno acciones como hacer un ejercicio, ver un mensaje en el foro, etc. en base a acciones pasadas de otros compañeros que él todavía no ha realizado. Consiste en un *plugin* que genera una ventana emergente cuando el estudiante selecciona un enlace o una acción, en la que se muestra la recomendación. Para generar las recomendaciones se usan reglas de asociación que tratarán de buscar eventos, típicamente, relacionados con el que acaba de hacer el estudiante, para proponérselos. Como datos de entrada, considera la información de los *log* genéricos de la plataforma que después debe convertir a acciones conocidas como acceso a un test, nuevo mensaje en el foro, etc. algo que ya nos proporciona directamente *Moodle*.

El – Halees [51] utiliza los datos de los estudiantes de la base de datos de *Moodle* para analizar su comportamiento utilizando técnicas como reglas de asociación, clasificación, *clustering* o detección de *outliers*. Para ello, utiliza datos de 151 estudiantes (datos personales, registros académicos, registros del curso y datos de los registros de *Moodle*) para obtener atributos como la asistencia, horas de dedicación, el GPA (*Grade Point Average*), recursos electrónicos, notas parciales y notas finales. Tras un pre – procesado de discretización de los valores numéricos, realiza cuatro análisis:

- Usa reglas de asociación para obtener reglas similares a las de otros estudios: si la asistencia a clase es buena, realiza los ejercicios propuestos en la plataforma y las notas parciales son buenas entonces la calificación final será excelente.
- Cómo método de clasificación utiliza C4.5 para clasificar alumnos y poder predecir la nota que tendrán en base a sus trabajos realizados.
- Utiliza *clustering* EM para agrupar estudiantes en base a su rendimiento. Con sus datos de partida consigue distinguir cinco *clusters* pero no indica por qué se caracterizan los estudiantes de cada uno, ni qué métrica toma para medir el rendimiento.
- En este caso, no elimina los *outliers* de los datos en la etapa de pre – procesado, sino que aplicará una técnica de detección de *outliers* para poder encontrar esos datos anómalos y analizarlos o resolverlos.

En la detección de *outliers* mediante *clustering* se centran Ajith, Say y Tejaswi [26]. A partir de datos relacionados con la asistencia a clase, nota de test realizados en clase, seminarios y tareas obtiene las variables de la Tabla 1.

Variable	Descripción	Posibles valores
PR	*Previous Results*	{Distinction> 70%, First > 60% & <70%, Second >50 & <60%, Third >40, & <50%, Fail < 40%}
MTR	*Mid Term Results*	{Poor <40 , Average >40 & <60, Good >60 & < 70, Best >70}
LW P	*Lab Work and Performance*	{Poor, Average, Good}
TS	*Technology Standards*	{Poor , Average, Good}
ASS	*Assignment*	{Yes, No}
SP	*Seminor Performance*	{Yes, No}
REG	*Regularity to class*	{Regular, Irregular}
PUN	*Puntuality*	{Yes, No}
OAP	*Overall Performance*	{Poor , Average, Good}
FE	*Final examination*	{Distinction >70, First > 60% & <70% Second >50 & <60% Third >40 & <50% Fail < 40%}

Tabla 1. Variables utilizadas en [26]

Aplica estos datos a un algoritmo de *clustering* (no especifica cuál) y agrupa estudiantes atendiendo a las variables anteriores. Por ejemplo, en función de su rendimiento medio se obtendrían tres *clusters*, como los mostrados en la Figura 21.

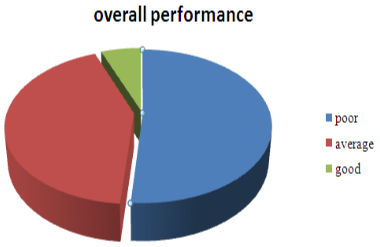


Figura 21. Clasificación según OAP en [26]

Obviamente lo interesante no es la agrupación, sino la detección de *outliers*, de aquellos puntos de datos que están lejos del resto de datos. En este caso, va a analizar los puntos de cada *cluster* para encontrar si existe alguno que esté bastante alejado del punto medio (denominado *centroide* y calculado como el valor medio de todos los elementos que componen el *cluster*). Por ejemplo, de entre los tres *clusters* anteriores, dentro del *cluster* GOOD ha encontrado un *outlier* (Figura 22). Para "separar" los datos normales de los *outliers* y mostrarlo en un histograma, aplica un análisis estadístico univariante.

Figura 22. Detección de *outliers* en [26]

En un ejemplo más completo, ha clasificado los estudiantes en base a su nota media parcial (MTR) y dentro de cada una de las cuatro categorías que resultan, en función de su rendimiento total (OAP). Ha encontrado, por ejemplo (ver Figura 23), que no es común que un estudiante que tenga muy buena nota parcial, tenga un rendimiento total medio, y por lo tanto considera que esto es una anomalía en los datos.

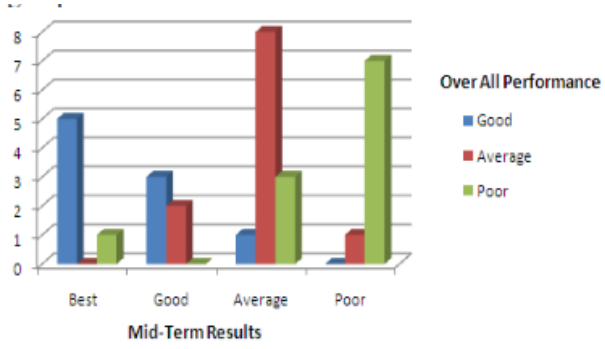


Figura 23. *Clustering* y detección de *outliers* en [26]

Morris, Finnegan y Wu [52] tienen por objetivo analizar las diferencias en la participación de los estudiantes en los cursos mediante un análisis estadístico. Usan los registros de accesos de 423 estudiantes en 3 cursos para comprobar qué diferencias de comportamiento ha habido entre aquellos que han terminado el curso aprobado (*successful completers* con nota A, B o C), suspendido (*non successful completers* con nota D, F o incompleta) y aquellos que lo abandonaron (*withdrawers*). Analizan la participación del estudiante en base a:

- Visitas al contenido del curso.
- Visitas a discusiones.
- Creaciones de nuevos *post* en el foro.
- Respuestas en el foro.

Una participación medida en base a:

- Variables frecuenciales:
    - Número de páginas de contenidos visitadas.
    - Número de discusiones en los foros leídas.
    - Número de *post* originales.
    - Número de *post* seguidos.
- Variables de duración:
    - Segundos dedicados a ver el contenido de las páginas.
    - Segundos dedicados en leer discusiones en los foros.
    - Segundos dedicados en crear *post* originales.
    - Segundos dedicados en crear *post* en discusiones ya existentes.

Concluye que la actividad de quien termina el curso es mayor que quien lo abandona y que se puede encontrar relación entre actividad y nota final.

Variable	Description	Possible Values
PSM	*Previous Semester Marks*	{First > 60%, Second >45 & <60% Third >36 & <45% Fail < 36%}
CTG	*Class Test Grade*	{Poor , Average, Good}
SEM	*Seminar Performance*	{Poor , Average, Good}
SS	*Assignment*	{Yes, No}
GP	*General Proficiency*	{Yes, No}
ATT	*Attendance*	{Poor , Average, Good}
LW	*Lab Work*	{Yes, No}
ESM	*End Semerter Marks*	{First > 60%, Second >45 & <60% Third >36 & <45% Fail < 36%}

Tabla 2. Datos utilizados en [24]

Baradwaj y Pal [24] utilizan clasificación mediante árboles de decisión para evaluar el rendimiento de los estudiantes. Utilizan los datos de la Tabla 2 y obtienen el conjunto de reglas IF-THEN-ELSE de la Figura 24.

IF PSM = 'First' AND ATT = 'Good' AND CTG = 'Good' or 'Average' THEN ESM = First
IF PSM = 'First' AND CTG = 'Good' AND ATT = "Good' OR 'Average' THEN ESM = 'First'
IF PSM = 'Second' AND ATT = 'Good' AND ASS = 'Yes' THEN ESM = 'First'
IF PSM = 'Second' AND CTG = 'Average' AND LW = 'Yes' THEN ESM = 'Second'
IF PSM = 'Third' AND CTG = 'Good' OR 'Average' AND ATT = "Good' OR 'Average' THEN PSM = 'Second'
IF PSM = 'Third' AND ASS = 'No' AND ATT = 'Average' THEN PSM = 'Third'
IF PSM = 'Fail' AND CTG = 'Poor' AND ATT = 'Poor' THEN PSM = 'Fail'

Figura 24. Árbol de decisión obtenido en [24]

Chandra y Nandhini [53] presentan un método para identificar estudiantes con patrones de suspenso. Después usan estos patrones para recomendar planes académicos que solucionen la situación. Este es un estudio bastante interesante desde el punto de vista educativo. No se indican los datos que se necesitan para ello, pero parece que serían la lista de asignaturas suspensas de cada estudiante, ya que los resultados que se obtienen son reglas de asociación que indican qué asignaturas es probable que suspenda un alumno, dada una lista de asignaturas ya suspensas, como se ve en la Figura 25.

RuleNO	Rule	Rule Support	Confidence
1	MAT122,ECO111 => CIS121	0.06	0.6
2	CIS111 => CSC212,ECO111	0.06	0.6
4	CIS121,MAT122 => ECO111	0.06	1
5	ACC111 => CHM111,CIS121	0.06	1
6	BUS211,ECO111 => MAT122	0.06	0.6
7	CIS121,BUS211 => CHM111	0.06	0.75
8	ACC111,BUS211 => CSM211	0.1	0.83

Figura 25. Reglas de asociación obtenidas en [53]

Los resultados de este estudio pueden ser aprovechados para realizar algo parecido a Vialardi y otros [54], los cuales proponen un sistema que tiene por objetivo recomendar al estudiante la matriculación o no en determinadas asignaturas en base a la experiencia con otros alumnos en su misma situación (principalmente, mismo expediente académico).

Para saber si un estudiante debe matricularse en un conjunto de asignaturas analiza estudiantes en situación académica similar que sí se matricularon. Con ellos genera una tabla en la que aparecerá una entrada por cada pareja estudiante – asignatura. Es decir, que si de las N signaturas en las que se quiere matricular el nuevo estudiante, el estudiante de prueba se matriculó en M, aparecerán M entradas para este estudiante de prueba. Cada entrada tendrá los siguientes atributos:

- Número de asignaturas matriculadas simultáneamente.
- Nombre del curso.
- Nota.
- Media académica del estudiante hasta antes de matricularse en esta asignatura.

Con todos estos datos va a predecir si el nuevo estudiante aprobará o suspenderá cada asignatura y, en base a estos resultados, recomendarle o no la matriculación. La predicción se realiza mediante técnicas de clasificación.

Por último, Cetintas, Si, Xin y Hord [55] proponen un método de detección de comportamiento *off-task* (básicamente como acciones distractoras de los alumnos cuando usan *Moodle*) que en principio no nos es de interés ya que proponen un observador que va anotando las acciones distractoras del alumno. Lo interesante de este artículo está en tres conceptos que pueden ser tenidos en cuenta en cualquier otro estudio:

- No sirve de nada considerar por separado las métricas de tiempo en hacer una tarea y rendimiento. Un alumno puede tardar muy poco pero obtener un mal resultado, por no haberse molestado; o un alumno no por obtener la mejor nota tiene que ser el mejor, porque puede haber tardado demasiado tiempo, este alumno puede que sea el mejor, pero no es eficiente. Es interesante valorar las capacidades de los estudiantes en base a la relación entre el resultado obtenido y el tiempo necesario para ello.
- Para cada estudiante no solo puede ser útil obtener métricas absolutas, sino también relativas (en relación a lo que han hecho el resto de compañeros).
- También puede ser interesante personalizar las métricas comparándolas con esos mismos valores para este estudiante, en otras situaciones. Por ejemplo, si es el tiempo

medio de realización de una actividad, compararla con el tiempo medio acumulado en todas las actividades anteriores para ver su evolución.

Para finalizar, indicar que los estudios aquí enumerados no se centran únicamente en *Moodle*. En aquellos trabajos en los que se no se ha especificado, la plataforma *e – learning* usada no es Moodle. Se ha ampliado la búsqueda para poder obtener mayor diversidad de resultados. Por otro lado, nos vamos a centrar en aquellas categorías de estudios con más relevancia en el sentido de más implementadas, aunque también nos podríamos plantear centrarnos en aquellos campos en los que se ha investigado menos. Debido a que este es el primer trabajo realizado en este campo de análisis por el grupo de investigación, se va a optar por empezar con aquellos estudios para los que podamos obtener mayor información para establecer las bases de futuros trabajos.

A lo largo del Capítulo 3 se detallarán los estudios que se han intentado realizar, a partir de la información extraída de este apartado, así como sus resultados.

2.2. *BigData*

En este segundo apartado abordaremos las distintas posibilidades que tenemos para realizar *BigData*.

2.2.1. Paradigmas de programación

El paradigma de programación escogido nos sienta las bases de todo el trabajo que vayamos a desarrollar. En función de él, las operaciones que queramos realizar deberán codificarse de una u otra forma, incluyendo el lenguaje de programación necesario para ello. Pero no solo nos fija cómo debemos programas las tareas, sino también cómo estas deben abstraerse y cómo, una vez definido el trabajo, este se ejecuta en la infraestructura correspondiente. En el contexto de soluciones *BigData* se utiliza paradigmas de computación en paralelo.

Sobre cualquiera de los paradigmas de programación que aquí se presentan se hará una breve introducción acerca de sus fundamentos. Más adelante, se detallará el paradigma *MapReduce* escogido para realizar el trabajo.

2.2.1.1. *MapReduce*

MapReduce [56]–[59] es un algoritmo de computación paralela sobre grandes cantidades de datos desarrollado por *Google*. Recibe su nombre de los dos métodos de los que se compone: *Map* y *Reduce*. Cuando el algoritmo recibe un nuevo trabajo con una serie de datos de entrada, divide estos datos en varias partes y con ellas genera unos pequeños sub – trabajos que reparte por los nodos disponibles para realizar el trabajo. Esto da lugar a la ejecución de varias tareas *Map* en paralelo, cada una de las cuales actúa sobre un pequeño subconjunto del total de datos a tratar, de forma independiente al resto de tareas. De esta forma, cada operación *Map* da lugar a un determinado resultado, en función del procesamiento que se pretenda realizar. Todos estos resultados pasan a una fase de operación *Reduce,* que es una operación de tipo agregación ya que se encargará de combinar todos los resultados de las distintas operaciones *Map* que se hayan ejecutado. Este principio de funcionamiento se esquematiza en la Figura 26. Más adelante se explicará más en detalle.

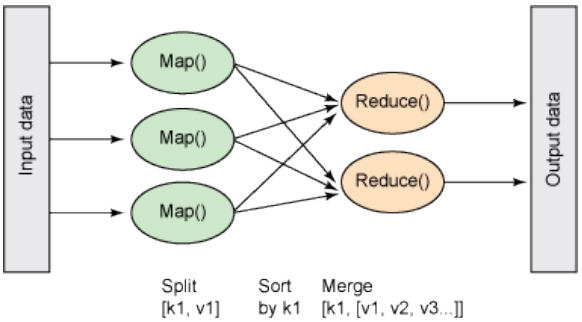


Figura 26. Esquema de funcionamiento de MapReduce. Fuente: [60]

Para llevar a cabo este comportamiento, la estructura contará con dos tipos de componentes diferenciados: un nodo maestro o central denominado *Master Node* y otros tantos nodos esclavos o trabajadores denominados *Worker Nodes*.

2.2.1.2. *Dryad*

Dryad [59], [61], [62] fue un proyecto de *Microsoft* con el objetivo de generar un escenario de computación distribuida de alto rendimiento. Las aplicaciones desarrolladas siguiendo este principio se modelan mediante un gráfico acíclico directo (*directed acyclic graph*, DAG), es decir, un flujo de tareas en el que no es posible que se generen bucles. Estos grafos están formados por vértices, que representan programas a ejecutar, y canales, el medio de comunicación entre programas (ficheros, tuberías *TCP*, memorias *FIFO*). En la Figura 27 se representa la estructura de un trabajo *Dryad*.

Poco más merece hablar sobre este paradigma ya que en el año 2011 *Microsoft* abandonó este proyecto para centrarse en *Apache Hadoop* y, por lo tanto, *MapReduce* [63].

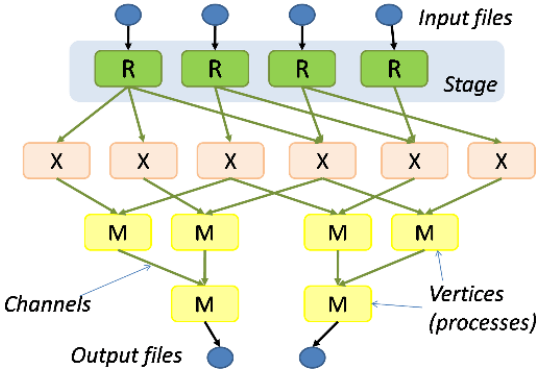


Figura 27. Estructura de un trabajo *Dryad*. Fuente: [61]

2.2.1.3. *Bulk Synchronous Parallel (BSP)*

BSP es un modelo de programación de procesamiento gráfico. Fue desarrollado a principios de la década de 1990. Se define como la combinación de tres atributos [64]:

- Una serie de componentes, cada uno de los cuales se encarga de funciones de procesado y memoria.

- Un *router,* que se encarga de entregar mensajes entre componentes y, por lo tanto, permite la comunicación entre ellos.

- Herramientas de sincronización.

La característica de sincronismo viene facilitada por estas últimas herramientas. El trabajo a realizar se divide en una secuencia de pasos denominados *supersteps* de una determinada duración fija. Cuando se inicia un nuevo *superstep,* a cada uno de los componentes se les asigna una tarea en la que combinará datos propios y mensajes recibidos. Cuando se consume el tiempo correspondiente al *superstep* se comprueba si todos los componentes han terminado su tarea. Si es así, se procede con el siguiente paso, en caso contrario, el siguiente paso estará destinado a terminar las tareas de los componentes que no se han podido completar aún.

Basándose en este modelo se desarrollaron paradigmas como *Pregel* [65] de *Google* o proyectos como *Apache Hama* [66], centrado en análisis *BigData* usando *BSP* como modelo de computación.

2.2.1.4. *GraphLab*

GraphLab [67]–[71] también se puede encuadrar dentro de los paradigmas de procesamiento basado en grafos aunque usa los principios de *MapReduce*. Surge, no tanto como alternativa a *MapReduce* si no como intento de poder realizar ciertos procesamientos para los cuales *MapReduce* no está optimizado. En concreto, su principal objetivo es conseguir una implementación que permita ejecutar fácil, eficiente y correctamente algoritmos de aprendizaje automático (*machine learning*) en paralelo.

En este modelo la función *Map* se sustituye por operaciones *Update*. Son similares, pero en este caso cada función *Update* puede leer y modificar conjuntos solapados de datos, es decir, las funciones *Update* no deben ser obligatoriamente independientes, como las operaciones *Map*. La operación análoga a *Reduce* se denomina *Sync*, la cual se encargará de realizar operaciones de reducción/combinación mientras otras tareas están en ejecución. También puede hacer frente a múltiples registros simultáneamente.

Al ser un modelo basado en grafo, toda la definición del trabajo se realiza mediante un grafo compuesto por vértices (*vertex*) y relaciones entre ellos (*edges*) en el cual se especifican los datos asignando puntos de memoria y en función del cual se controla la interacción entre funciones *Update*.

2.2.1.5. *Otros*

- *Piccolo* [59] es un marco de trabajo diseñado para poder desarrollar fácilmente aplicaciones distribuidas eficientes para ejecución paralela y en memoria usando varias máquinas. La mayoría de las alternativas existentes presentan un mismo problema: no están convenientemente diseñadas para poder realizar computación en memoria, sin escribir en disco, ya que no exponen el estado global de los trabajos, algo en lo que se centra *Piccolo* con el objetivo de poder abarcar aplicaciones que requieren compartir el estado intermedio de las operaciones y aplicación *on – lines* que requieren acceso para modificar su estado.

- *Massive Parallel Processing* (MPP) [59] se centra en el uso de un gran número de procesadores en el mismo o distintos ordenadores para ejecutar tareas de computación coordinadas y simultáneas.
- Redes *peer – to – peer* [72] compuestas por millones de máquinas conectadas en red. Cada uno de los pares actúan tanto como servidor como cliente de recursos. Típicamente se usa MPI como método de comunicación entre pares.
- *Clusters* de computación de alto rendimiento (*High Performance Computing*, HCP, *clusters*), compuestos por supercomputadores [72].
- Usar *CPU* multinúcleo [72].
- Procesamiento en *GPU* [72].
- Usar *FPGAs* [72].

2.2.1.6. *Conclusión*

A lo largo de esta sección se ha podido observar la existencia de distintas alternativas en cuanto al paradigma de programación a usar para empezar a desarrollar un trabajo de *BigData*. Realmente son modelos de procesamiento de datos en paralelo, que se pueden utilizar en cualquier otro contexto. No se ha incidido demasiado en cada una de las alternativas, dado que de antemano se ha optado por utilizar *MapReduce*, ya que, a pesar de tener ciertos inconvenientes, como se indicará en la siguiente sección, es uno de los algoritmos más extendidos. Por otro lado, como se sabe cuáles son sus puntos débiles, se han desarrollado diversas versiones o mejoras de *MapReduce* para poder adaptarlo a situaciones en las que el modelo original no está optimizado.

Para finalizar este apartado indicar que en páginas *web* como [73]–[75] se puede encontrar listas con estas y otras alternativas a *MapReduce*.

2.2.2. Plataformas y variantes de *MapReduce*

Tras haber analizado las distintas alternativas de modelos de programación a utilizar hemos optado por escoger *MapReduce*. Pero con esta elección no acaban los problemas. Es de sobra conocido que *MapReduce* tiene una serie de limitaciones que hacen que su funcionamiento no sea eficiente en ciertos casos. En este apartado trataremos con distintas "versiones" existentes de *MapReduce*.

Apache Hadoop es la implementación de código abierto más conocida de *MapReduce* (en su versión "original"). *Hadoop* es muy útil en gran cantidad de casos, sobre todo en aquellos en que los datos a procesar se pueden partir en trozos independientes, para poder tratarlos en operaciones *Map* independientes, y después unir los resultados de estas funciones para dar un único resultado final. De esta forma se puede paralelizar el trabajo a lo largo de un conjunto de máquinas. Obviamente, existe una serie de casos en los que *Hadoop* no es apropiado en el sentido de ineficiencia porque el trabajo, aun pudiéndose llevar a cabo, requiere tanto tiempo que no se puede considerar que se haya paralelizado. Entre estos casos se podrían citar:

- Dada la definición del algoritmo *MapReduce*, todos aquellos problemas en los que los datos no se puedan dividir en partes independientes (bien porque la computación debe realizarse sobre todos los datos simultáneamente o porque se requiera intercambiar datos intermedios entre procesos) no serán tratados eficientemente por el paradigma.

- *Hadoop* no es la opción más adecuada para tratar con algoritmos de aprendizaje automático iterativo (*machine learning*). Esta cuestión ya se introdujó en la sección anterior. Es una de las limitaciones más fuertes que tiene el paradigma y para la cual están surgiendo alternativas. Atendiendo a [76] *Hadoop/MapReduce* es adecuado para la ejecución de algoritmos iterativos simples donde el algoritmo se puede expresar como la ejecución de un único modelo *MapReduce* o la ejecución secuencial de varios de estos modelos, pero no está eficientemente diseñado para afrontar algoritmos que solo se pueden expresar de tal forma que cada iteración es un único modelo *MapReduce* o está compuesta por varios modelos *MapReduce* (como por ejemplo, el algoritmo del gradiente descendente, un algoritmo típico de aprendizaje automático.).

A pesar de que puedan parecer problemáticas estas limitaciones, *Apache Hadoop* es la infraestructura más usada en problemas *BigData*. Aun así, no solamente han aparecido algoritmos alternativos a *MapReduce* que tratan de paliar estas contraprestaciones (como se abordó en el apartado anterior), sino también mejoras o adaptaciones de *MapReduce* que ofrecen más funcionalidad o una forma distinta de ver el algoritmo.

Es bastante común mezclar estas "otras versiones" de *MapReduce* como alternativas a *MapReduce*. En este documento se ha tratado de ver como algo complementario y no alternativo y por ello se ha planteado como un apartado aparte. De esta forma, en los sitios *web* citados anteriormente [73]–[75] también se puede encontrar referencias a algunas y otras más "versiones" que aquí se tratan.

También es un poco complicado tratar todo lo que viene a continuación como versiones de *MapReduce*. Algunas de estas alternativas son directamente plataformas de trabajo que usan *MapReduce* pero con alguna ligera modificación para paliar ciertas desventajas o proporcionar alguna característica nueva.

En primer lugar, podemos encontrarnos "nuevas" formas de implementar el algoritmo, por ejemplo, ***BashReduce*** [74], [77] es la versión *MapReduce* implementada mediante comandos *UNIX*, y por lo tanto consiste en un *script*. La versión original de *MapReduce* implicaría la construcción de programas siguiendo algún lenguaje de alto nivel como *C++* o *Java* si lo queremos implementar sobre *Hadoop*, algo que requeriría seguir la estructura típica de proyectos de este tipo. El problema que tiene *BashReduce* es el sistema de ficheros. *MapReduce* trabaja sobre un sistema de ficheros como pieza fundamental. Como está orientado a programación en paralelo a lo largo de un conjunto de máquinas, este sistema de ficheros es distribuido siguiendo un determinado modelo, por ejemplo para *Hadoop* se usa *HDFS*, como se verá más adelante. Trabajando con comandos *UNIX* el sistema de ficheros es el que corresponda a la máquina donde se están ejecutando y, por tanto, no está distribuido, de tal forma que, al tratar de distribuir el trabajo entre las máquinas esclavas o trabajadoras también tiene que distribuir los ficheros con los datos necesarios (mediante *SSH*) con la consiguiente carga de transferencia y los problemas de falta de tolerancia a fallos de los sistemas de ficheros no distribuidos. Comparado con *Hadoop*, es más sencillo y rápido, pero tiene falta de tolerancia a fallos, por usar un sistema de ficheros no distribuido, y no permite usar todos los comandos *UNIX*. Es por ello, que se le considera una mera herramienta a no tan alto nivel como *MapReduce*.

MapIterativeReduce [78] es un paradigma alternativo que extiende el modelo de programación de *MapReduce* para dar mejor soporte a aquellas aplicaciones que requieren de fun-

ciones *Reduce* con mucha carga de trabajo. Para ello, a continuación de la función *Map* clásica se introduce un función *Reduce* iterativa. Con esta idea, una función *Reduce* puede que tenga que combinar salidas de datos de funciones *Map* pero también de otras funciones *Reduce* presentes en iteraciones anteriores.

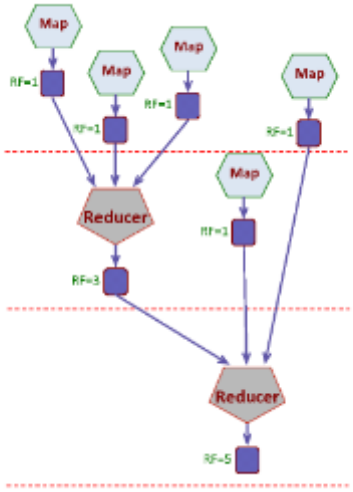


Figura 28. Esquema de funcionamiento de *MapIterativeReduce*. Se puede comparar con la Figura 26 donde se representa el esquema de funcionamiento de *MapReduce*. Fuente: [78]

Map – Reduce – Merge [79] consiste en una mejora de *MapReduce* añadiendo una nueva fase (*Merge*) con la que se puede unir datos particionados y ordenados por las funciones *Map* y *Reduce*. Todavía no se ha comentado, pero en los correspondientes esquemas de *MapReduce* se ha podido observar que el trabajo *MapReduce* no produce una única salida, sino una por cada operación *Reduce* que se lleva a cabo, normalmente no suele ser necesario juntar estos resultados. *Map – Reduce – Merge* permite unir ciertos resultados según la lógica escogida por el usuario cuando se manejan varios conjuntos de *mappers* y *reducers*, como en la Figura 29.

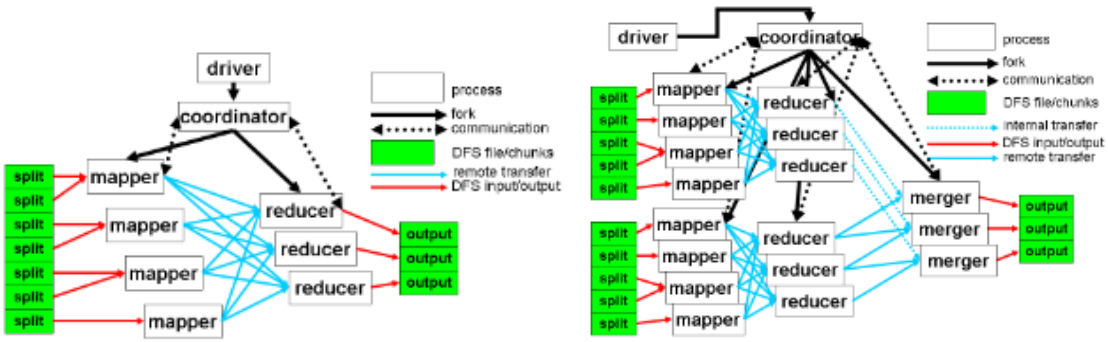


Figura 29. Comparativa entre *MapReduce* (izquierda) y *Map – Reduce – Merge* (derecha). Fuente: [79]

Alternativas directas a *Apache Hadoop* podrían considerarse *Disco Project* y *Cloud MapReduce*. ***Disco Project*** DISCO [80] consiste en un marco de trabajo para computación distribuida basado en el paradigma *MapReduce* desarrollado inicialmente por *Nokia*. La diferencia con *Hadoop* es que los trabajos *MapReduce* se escriben en *Python* y no en *Java*. Escrito en *Erlang*, distribuye y replica los datos, haciendo uso de un sistema de ficheros estándar, lo que le resta tolerancia a fallos. Por lo demás, proporciona una eficiente herramienta de planificación de trabajos y, se dice [74], elimina los peores aspectos de *Hadoop*. En el mismo sentido puede moverse ***Cloud MapReduce*** [81], una implementación de *MapReduce* sobre el sistema operativo *Amazon Cloud*. El hecho de no implementarlo sobre un SO tradicional, según sus

autores [82], trae una serie de ventajas como ser más rápido que *Hadoop* (aunque puntualizan que en según qué ocasiones), ser más escalable al no existir un punto de embotellamiento o ser más simple en construcción en cuanto a líneas de código necesarias.

Twister [83], [84] y *HaLoop* [85] surgen como alternativas para implementar *MapReduce* de forma iterativa. ***HaLoop*** directamente se define como una versión modificada de *Hadoop*/*MapReduce* para dar servicio a aplicaciones iterativas. De una forma similar se podría definir ***Twister*** , sin ser una versión modificada de *Hadoop*.

Finalmente, pero no menos importantes, son las plataformas *Apache Spark* y *Apache Storm*. ***Apache Spark*** [86] es una plataforma de computación *BigData* que implementa una versión ligeramente modificada de *MapReduce* con la cual se consigue ejecutar aplicaciones en menor tiempo. Para ello, en lugar de realizar escrituras/lecturas en disco durante la ejecución de la aplicación, guarda la información correspondiente en memoria, es por ello que es un marco de trabajo más exigente en cuanto a especificaciones técnicas. Además, esta nueva implementación, es una plataforma adecuada para implementar algoritmos de aprendizaje automático (*machine learning*). Por su parte, ***Apache Storm*** [87], es desarrollado por *Twitter* con el objetivo de poder realizar análisis de datos en tiempo real (es considerado como el *Hadoop* en tiempo real), lo cual también se hace adecuado para implementar aprendizaje automático, como *Spark*.

Como conclusiones podemos extraer que *MapReduce* es uno de los algoritmos más utilizados dentro del análisis *BigData* a pesar de tener serias limitaciones. Esto puede ser debido a que se han desarrollado versiones o plataformas basadas en *MapReduce* que solventan estos problemas. De entre todas las posibles implementaciones que existen de *MapReduce*, la más conocida es *Apache Hadoop*, a pesar de que también puede llegar a tener sus inconvenientes. El hecho de que *Apache Hadoop/MapReduce* tenga sus problemas pero no hayan sido un obstáculo para su triunfo puede estar en la gran cantidad de herramientas adicionales que existen por detrás y que facilitan su uso o mejoran sus prestaciones. En este apartado se ha hablado de *Apache Spark* y *Apache Storm*, dos herramientas poderosas, prácticamente son plataformas independientes que se pueden integrar con *Hadoop* para suplir algunas de sus carencias. Es por ello que no se ha incidido mucho más en ellas, ya que el siguiente apartado estará destinado a abordar la arquitectura de la plataforma *Hadoop* y todas aquellas herramientas que se suelen integrar con ella para hacerla aún más poderosa.

Por otro lado, el análisis aquí realizado también ha sido una mera introducción a todas las variantes de *MapReduce* que nos podemos encontrar. Existen numerosos artículos que comparan cada una de las alternativas aquí citadas con *Hadoop* (ya que se considera como la referencia en *BigData* y *MapReduce*) y con otras opciones de modelo de programación indicadas en el Apartado 2.2.1.

2.2.3. *Apache Hadoop*/MapReduce

En este punto ya hemos escogido *Apache Hadoop* como plataforma para llevar a cabo nuestro análisis *BigData* sobre *Moodle*. Aunque podemos pensar que en este punto poco más hay que decidir, no es así.

Apache Hadoop por sí mismo no tiene mucho valor, lo adquiere al integrarse con otra serie de herramientas que añaden funcionalidades y facilidad de uso, como se verá a lo largo de esta sección. Una primera opción es optar por instalar manualmente cada uno de estos paquetes *software*. Otra opción es escoger una distribución de *Hadoop* ya preparada con todas estas

herramientas. El objetivo de este trabajo es desarrollar una serie de métodos *BigData* para aplicar análisis dentro de *Moodle*, obviamente no es importante el método que se use para conseguirlo. De esta forma, lo más sencillo para empezar a trabajar rápidamente en lo importante es optar por una distribución en la que ya tengamos todo preparado para comenzar a programar.

Son numerosas las empresas/compañías que comercializan tanto de forma gratuita como de pago, distribuciones *Hadoop*. Teniendo esto en mente, debemos escoger cual es aquella que más nos puede facilitar nuestro objetivo, para lo cual, en primer lugar, se ha realizado un análisis de las dos distribuciones *Hadoop* más extendidas: *Hortonworks* y *Cloudera*.

Una vez escogida la distribución que vamos a usar, se desarrolla la arquitectura *Hadoop* que sigue esa distribución, los paquetes *software* de los que se compone y sus funcionalidades.

2.2.3.1. *Distribuciones de Hadoop: Hortonworks vs Cloudera*

En el mercado de distribuciones *Hadoop* son tres los nombres predominantes: *Hortonworks* Hortonworks [88], *Cloudera* cloudera [89] y *MapR* MAPR [90] (Figura 30). De entre estas tres hemos optado por analizar las dos primeras. Son muchos los artículos o páginas *web* [91], [92] en las que se realizan comparativas entre ambas distribuciones. En nuestra opinión, unas comparativas pocos útiles ya que son pocas las diferencias que se pueden apreciar. Por ejemplo, en la Tabla 3 se muestra una comparativa entre ambas, extraída de [92] de abril del 2015. La única diferencia es la herramienta escogida para la monitorización de servicios y dispositivos (*Apache Ambari* vs *Cloudera Manager*) y el hecho de que *Hortonworks* no tiene herramientas de recuperación de desastres, mientras *Cloudera* sí. Estos son dos puntos que no nos interesan demasiado en este trabajo. En primer lugar, la herramienta de monitorización de servicios nos da igual lo avanzada que sea o demás; nos interesa poder lanzar, detener y configurar servicios; de igual forma con respecto a los dispositivos. Por otro lado, la recuperación ante desastres tampoco es relevante, ya que queremos una infraestructura para probar algoritmos, no para ponerlos en marcha y tenerlos constantemente operativos. De esta forma, esta primera comparativa no nos es mucho de ayuda. Otro punto destacable es que el *software Hortonworks* es 100% libre, mientras que *Cloudera* proporciona herramientas propias (por ejemplo, *Cloudera Manager*).

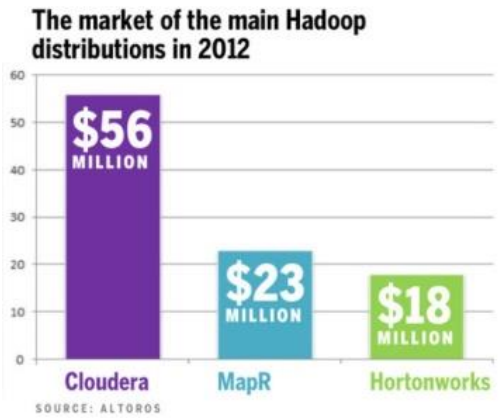


Figura 30. Comparativa comercial *Hortonworks – Cloudera – MapR*. Fuente: [91]

	Hortonworks	cloudera
	Rendimiento y escalabilidad	
Procesado de datos	Por lotes	Por lotes
Arquitectura de metadatos	Centralizada	Centralizada
Rendimiento *HBase*	Picos de latencia	Picos de latencia
Aplicaciones *NoSQL*	Aplicaciones por lotes	Aplicaciones por lotes
	Confianza	
Disponibilidad	Un único punto de recuperación	Un único punto de recuperación
Disponibilidad *MapReduce*	Reiniciar trabajos	Reiniciar trabajos
Actualización	Tiempo de inactividad planeado	Actualizaciones sucesivas
Replicación	Datos	Datos
Snapshots	Consistencia solo para ficheros cercanos	Consistencia solo para ficheros cercanos
Recuperación de desastres	No	Planificación de copia de ficheros
	Gestión	
Herramienta de gestión	*Apache Ambari*	*Cloudera Manager*
Soporte para volúmenes	No	No
Mapas de color, alarmas, alertas	Si	Si
Integración con API REST	Si	Si
Control datos/trabajos	No	No
	Acceso a los datos	
Sistema de ficheros	*HDFS* y *NFS* de solo lectura	*HDFS* y *NFS* de solo lectura
Entrada salida de ficheros	Solo anexar	Solo anexar
Seguridad: *ACLs*	Si	Si
Autenticación	*Kerberos*	*Kerberos*

Tabla 3. Comparativa distribuciones *Hortonworks* y *Cloudera* para *Hadoop*. Fuente: [92]

Viendo que las comparativas técnicas no nos permiten decidir entre ambas, nos vamos a centrar a analizarlas en la práctica. Anteriormente se ha comentado que lo interesante de las distribuciones existentes es que proporcionan un entorno en que tengamos disponibles todas aquellas **herramientas y paquetes *software*** que nos facilitan el uso de *Hadoop*. Nuestro siguiente paso es comparar qué funcionalidades presentan, como se presenta en la Tabla 4. Visto que presentan los mismos paquetes *software* integrados (al menos los que a simple vista nos parecen relevantes o necesarios para este trabajo), se compararon las versiones de los mismos. De esta comparativa se puede concluir que *Hortonworks* proporciona, por lo general, versiones más recientes de los paquetes *software*. La única diferencia entre ambas distribuciones está en el motor de ejecución de consultas en paralelo: *Apache Tez* en *Hortonworks* e *Impala* en *Cloudera*. *Cloudera Impala* es un motor de consultas *SQL* propio de *Cloudera* que solamente se ejecuta sobre *Hadoop*. *Cloudera* indica que es mucho mejor que el "genérico" *Hive* (disponible tanto en *Hortonworks* como *Cloudera*), ya que es mucho más rápido, ejecuta las mismas sentencias en menos tiempo (en algunas referencias se indica que solamente es más rápido que *Hive* en determinadas situaciones [93]). En la práctica hemos podido comprobar que así es, pero a menos de un problema de congestión de la máquina u otras cuestiones que desconocemos, observamos un comportamiento extraño, ya que la misma sentencia *SQL* ejecutada en *Hive-Hortonworks* y *Hive-Cloudera* difieren mucho en tiempo de ejecución, cuando

se supone que la base de ejecución es la misma, *Hive-Cloudera* tarda mucho más en ejecutar una simple consulta. Además, parece que dada la definición de *Impala* solo sería una alternativa a *Hive* por lo tanto parece extraño compararla con *Apache Tez*, como se realiza en la referencia [91]. *Apache Tez* es un entorno que permite agilizar trabajos *MapReduce* y es utilizado por distintas herramientas como *Hive* o *Pig*.

		Hortonworks HDP 2.3	cloudera CDH 5.4.2
Sistema de ficheros	*HDFS*	*Hadoop YARN* 2.7.1	*Hadoop YARN* 2.6.0
	Acceso *Web – API REST*	*WebHDFS*	*HttpFS*
MapReduce		*MapReduce2* 2.7.1	*MapReduce2* 2.6.0
Base de datos no relacional	*Apache* HBase	1.1.1	1.0.0
Servicio de Metadatos	*Apache HCatalog*	1.2.1	1.1.0
Script	*Apache Pig*	0.15.0	0.12.0
	DataFu	Versión desconocida	1.1.0
Acceso a los datos y consulta	*Apache Hive*	1.2.1	1.1.0
	Impala	-	2.2.0
Planificación de flujos de trabajo	*Apache Oozie*	4.2.0	4.1.0
Coordinación de *clusters*	*Apache ZooKeeper*	3.4.6	3.4.5
Transferencia con bases de datos relacionales	*Apache Sqoop*	1.4.6	1.4.5
Gestión de datos de *log*	*Apache Flume*	1.5.2	1.5.0
Algoritmos de aprendizaje automático	*Apache Mahout*	0.9.0 (no instalado de origen)	0.9
Interfaz *Hadoop*	*Hue*	2.6.1	3.7
Servicios en la nube	*Whirr*	-	0.9.0
Administración	*Apache Ambari*	2.1.0	-
	Cloudera Manager	-	CDH 5.4.2
Operaciones no MapReduce	*YARN*	*Hadoop YARN* 2.7.1	*Hadoop YARN* 2.6.0

Motor de ejecución de consultas en paralelo	*Apache Tez*	0.7.0	-
	Impala	-	2.2.0
Gestión de datos	*Apache Falcon*	0.6.1	-
Procesamiento en tiempo real	*Apache Storm*	0.10.0	-
Almacenamiento clave-valor	*Apache Accumulo*	1.7.0 (no instalado de origen)	-
Gobierno de meta-datos	*Apache Atlas*	0.5.0	-
Cola de mensajes	*Apache Kafka*	0.8.2	
Autenticación en red	*Apache Kerberos*	1.10.3 (no instalado de origen)	1.8.2
Autenticación	*Apache Knox*	0.6.0	-
	Apache Sentry	-	1.4.0
Seguridad	*Apache Ranger*	0.5.0 (no instalado de origen)	-
	Apache Sentry	-	1.4.0
Despliegue de aplicaciones distribuidas en *YARN*	*Apache Slider*	0.80.0	-
Procesamiento rápido de datos	*Apache Spark*	1.3.1	1.3.0
Búsqueda de datos en *HDFS*	*Apache Solr*	5.2.1	4.10.3
	Cloudera Search	-	1.0.0
Llamadas de procedimiento remoto	*Apache Avro*	-	1.7.6
Tuberías *MapReduce*	*Apache Crunch*	-	0.11.0
Mediador *Impala* – *YARN*	*Llama*	-	1.0.0
Almacenamiento en columnas	*Parquet*	-	1.5.0

Tabla 4. Comparación versiones en *Hortonworks* y *Cloudera*

Tras analizar las versiones, tratamos de instalar un entorno de prueba de cada una de las dos distribuciones para probarlas con algún ejemplo sencillo de los tutoriales y documentaciones de ambas. Para instalar el entorno ambas proporcionan ***sandboxes*** a instalar sobre entor-

nos de virtualización, algo muy interesante ya que evita la necesidad de instalar los entornos en máquinas reales y consiguientes inconvenientes asociados a ser entornos para la realización de pruebas. En cuanto a estos *sandboxes*, tanto *Hortonworks* como *Cloudera* proporcionan *sandboxes* instalables tanto en *VirtualBox* como *VMWare* y por lo tanto podemos escoger la que necesitemos según la herramienta de virtualización que usemos. Un *sandbox* no es más que una máquina virtual en la que se ha instalado por completo *Hadoop* junto con todas las herramientas auxiliares que se han mencionado anteriormente y se detallarán más adelante.

Visto que ambas proporcionan la misma facilidad en cuanto a *sandboxes*, procedemos a instalar una máquina virtual de cada una de ellas y probarlas. En primer lugar, la máquina virtual *sandbox* de *Hortonworks* no tiene entorno gráfico, se maneja únicamente por línea de comandos. La máquina virtual de la *sandbox* de *Cloudera* tiene un entorno gráfico dentro del cual manejar *Hadoop*. En opinión personal, sobre este punto, el hecho de que *Cloudera* tenga entorno gráfico no le proporciona ninguna ventaja.

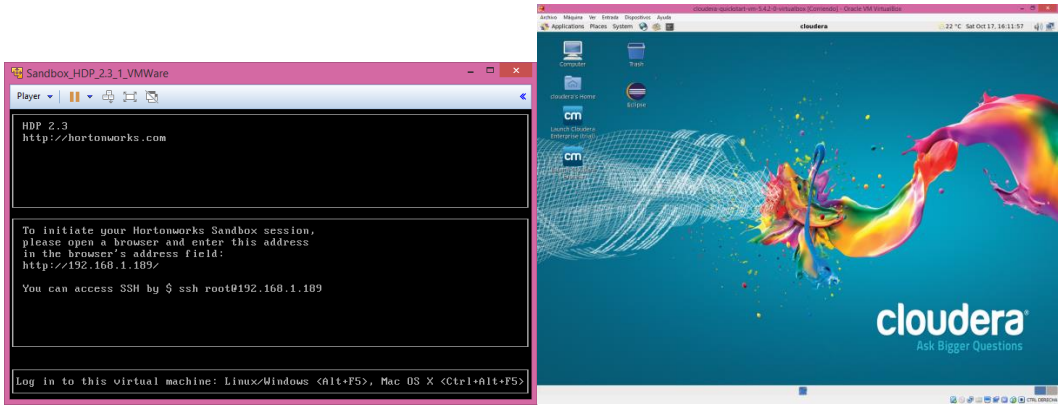


Figura 31. Comparativa máquinas virtuales *Hortonworks* (izquierda) vs *Cloudera* (derecha). La *sandbox* de *Hortonworks* no tiene entorno gráfico, toda gestión de la MV debe hacerse desde consola. *Cloudera* proporciona un entorno gráfico bastante desarrollado con todas las herramientas que se puedan necesitar

Las herramientas adicionales se pueden manejar por línea de comandos o por medio de una interfaz *web*. Para acceder a la interfaz *web* de *Hortonworks* se accede mediante un navegador *web* desde el sistema operativo anfitrión, usando la dirección IP proporcionada por la máquina virtual. Como la *sandbox* de *Cloudera* tiene entorno gráfico, se puede acceder desde un navegador *web* dentro de la misma máquina virtual o desde el sistema operativo anfitrión. Lo único ventajoso que se ha podido encontrar, en este sentido, de tener entorno gráfico en la máquina virtual es que el navegador pre – instalado cuenta con una serie de marcadores con las direcciones de acceso a los distintos servicios (Figura 32). En *Hortonworks* cuesta un poco más encontrarlas.

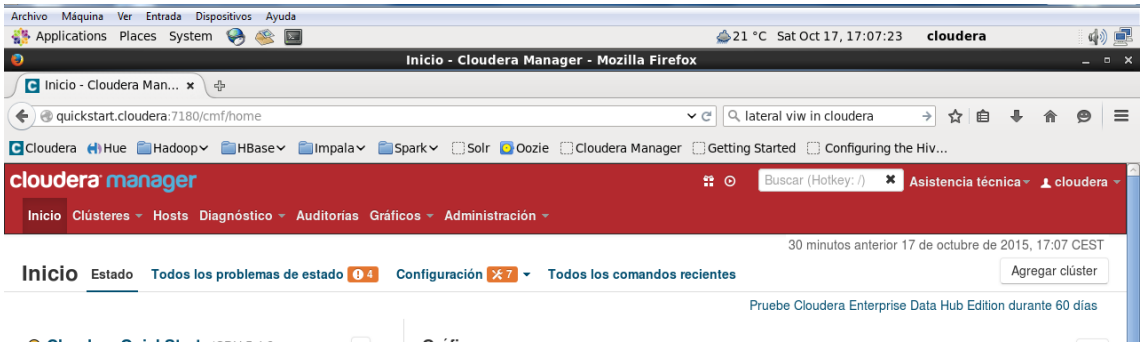


Figura 32. En el navegador *web* pre – instalado en la *sandbox* de *Cloudera* podemos encontrar marcadores hacia las distintas interfaces que nos permiten usar *Hadoop*

Otro punto que se ha tenido en cuenta es que se pueden cargar ficheros en el sistema de ficheros de *Hadoop* (*HDFS*) a partir de la interfaz *web* desde la que se accede al gestor de ficheros. Consideramos que es mucho más cómodo acceder a la interfaz *web* desde el navegador del sistema operativo anfitrión y tener en él los ficheros correspondientes que tener que transferirlos desde el sistema operativo anfitrión al virtual y de ahí a *HDFS*, por lo que pierde peso la necesidad de entorno gráfico. Aunque sí que es verdad que gestionar, no el entorno *Hadoop* sino el entorno de la máquina virtual es mucho más fácil si tenemos entorno gráfico.

En cuanto a estas **interfaces *web***, *Hortonworks* tiene dos: *Apache Ambari* y *Hue*; mientras que *Cloudera* también tiene otras dos: *Cloudera Manager* y *Hue*. *Apache Ambari* permite tanto monitorizar y gestionar servicios y nodos, como utilizar alguno de estos servicios con fines de análisis. *Cloudera Manager* solo permite monitorizar y gestionar. Comparando estas dos interfaces, es mucho más interesante *Cloudera Manager*, ya que presenta gran cantidad de gráficos para monitorizar el estado de los servicios lanzados, máquinas que conforman el *cluster Hadoop*, etc. *Ambari* no es tan potente en este aspecto, pero quizá es más fácil de entender y utilizar. En la Figura 33 pueden verse la página principal de *Apache Ambari* donde se muestra el estado del *cluster* en general. El detalle sobre el estado de un *host* se muestra en la Figura 34. En la Figura 35, se puede ver la página principal de *Cloudera Manager* y en la Figura 36 la monitorización de un *host* del *cluster*, con gráficos muy interesantes.

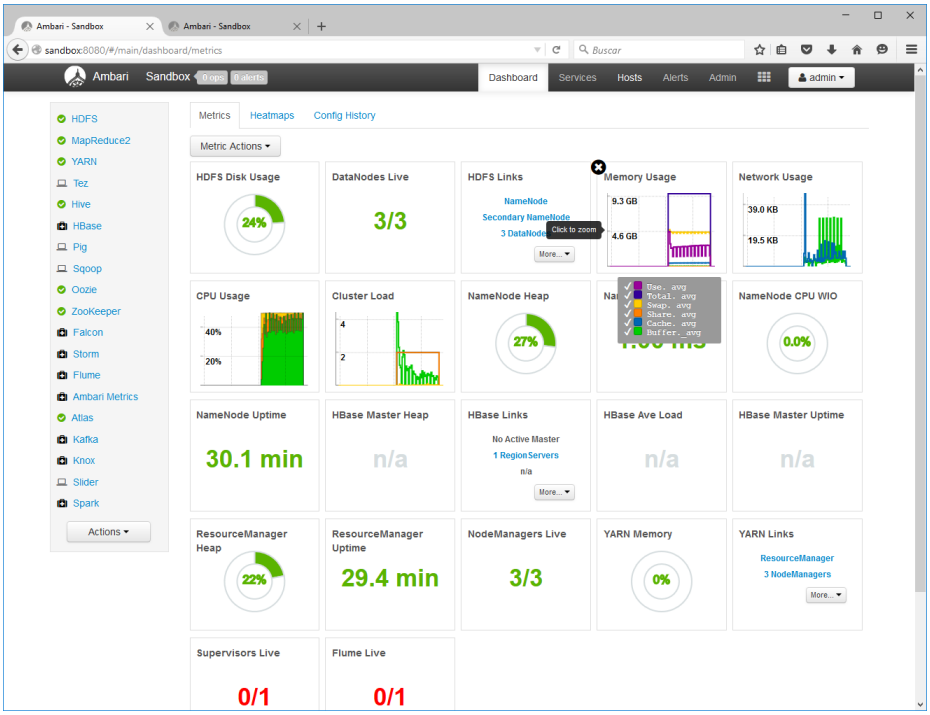


Figura 33. Página principal de *Apache Ambari*

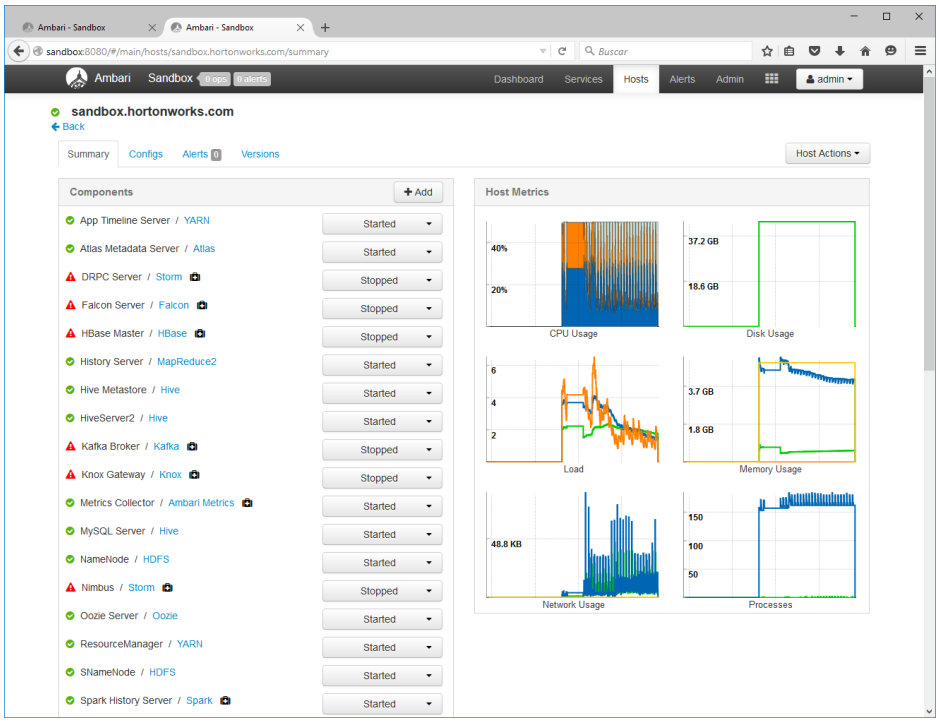


Figura 34. Página de estado de *host* en *Apache Ambari*

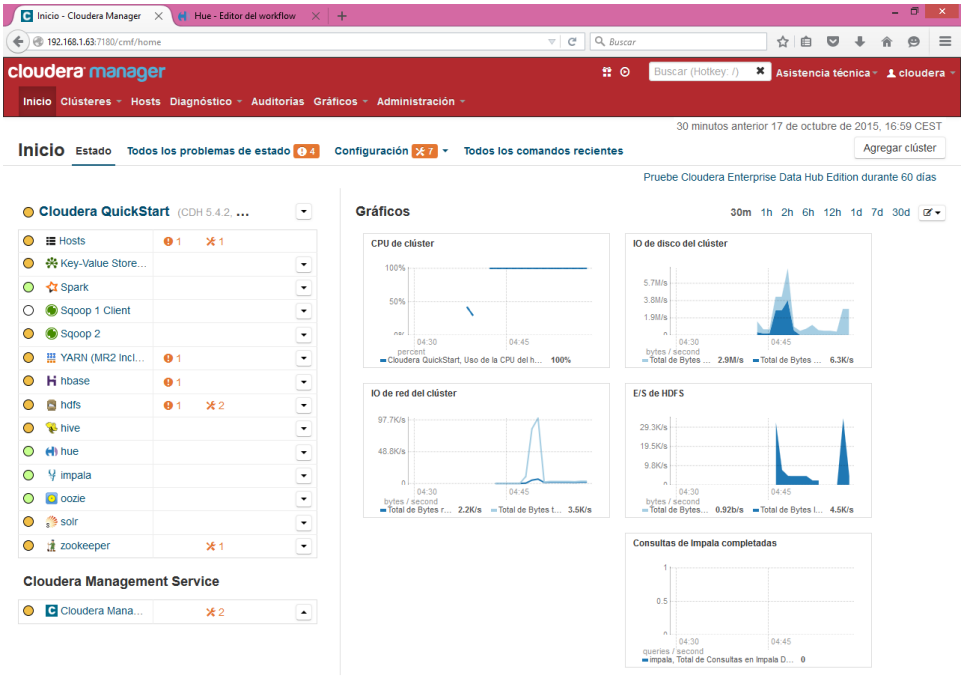


Figura 35. Página principal de *Cloudera Manager*

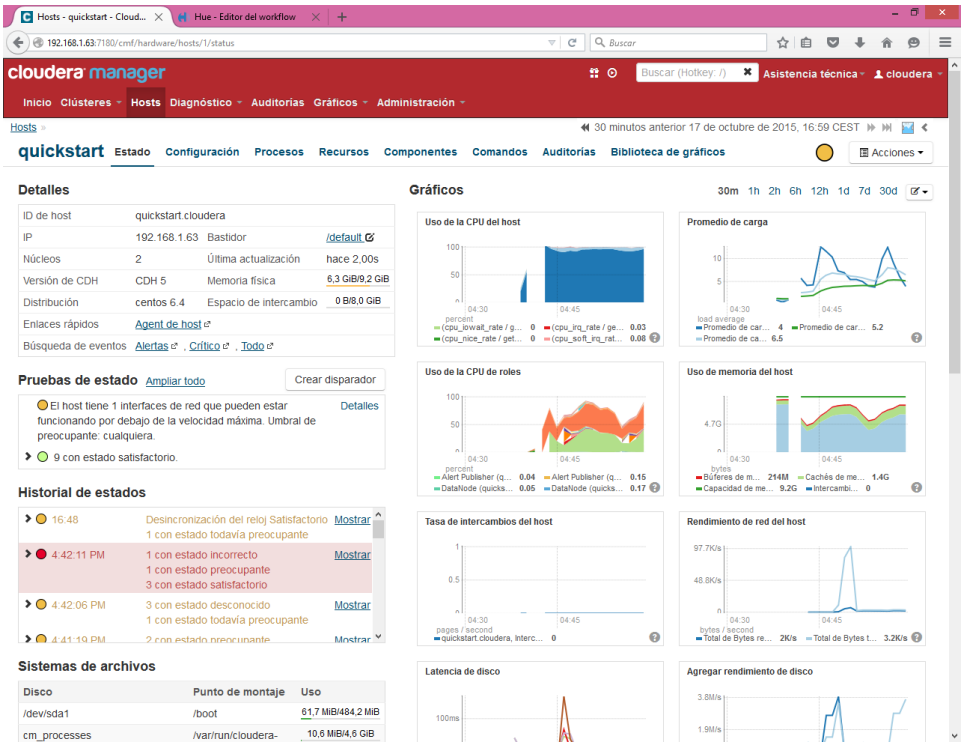


Figura 36. Página de estado de *host* en *Cloudera Manager*

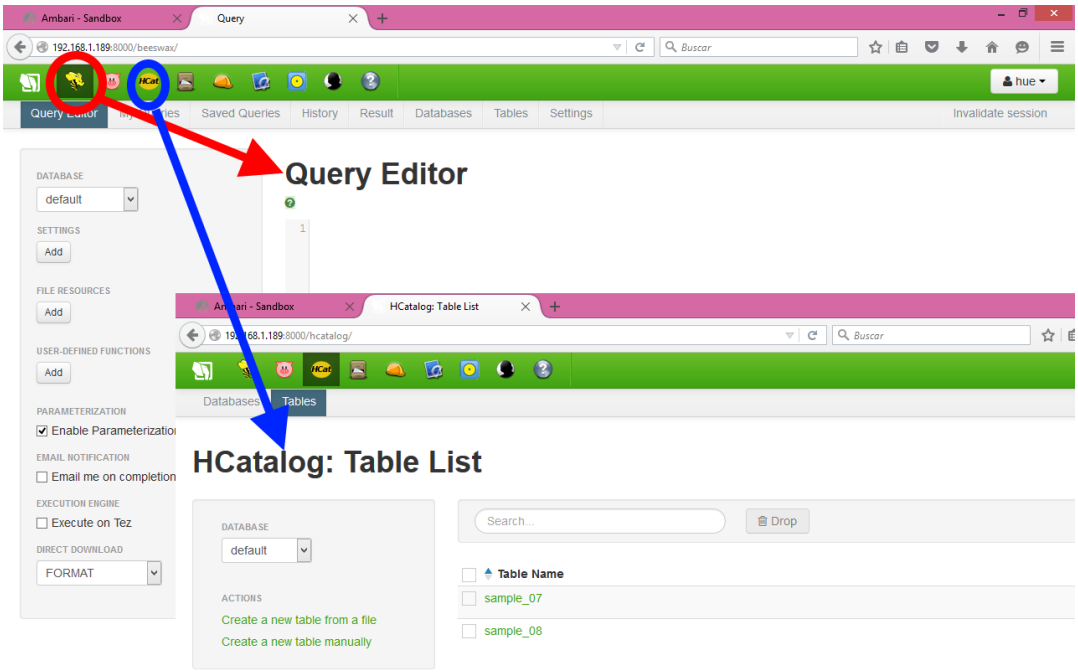


Figura 37. En *Hue* 2.6 (pre – instalado en la *sandbox* de *Hortonworks*) el terminal de consultas *Hive* y la navegación por las bases de datos, tablas y demás creadas se realiza a través de dos puntos distintos: Editor *Hive* y *HCatalog*, respectivamente

Hue es una interfaz que permite usar los paquetes *software* integrados alternativamente a ejecutarlos desde línea de comandos. La única diferencia entre ambas distribuciones es la versión de *Hue*. La versión en *Hortonworks* (*Hue* 2.6) es más antigua que en *Cloudera* (*Hue* 3.7). No se puede decir que cuanto más reciente sea mejor. Poniendo en práctica algunos ejemplos se ha puesto de manifiesto que algunas tareas son más fáciles de llevar a cabo en una u otra versión.

En nuestra opinión, una parte importante de estas interfaces es facilitar el uso de los paquetes integrados como *Hive*, *Pig* u *Oozie*, ya que consideramos estas las herramientas fundamentales que vamos a necesitar. Para manejar *Hive* parece más adecuada la interfaz de *Hue* 3.7 ya que presenta conjuntamente el terminal para lanzar nuevas consultas (*Hive*) y poder acceder al contenido (*HCatalog*), tablas y demás de la base de datos.

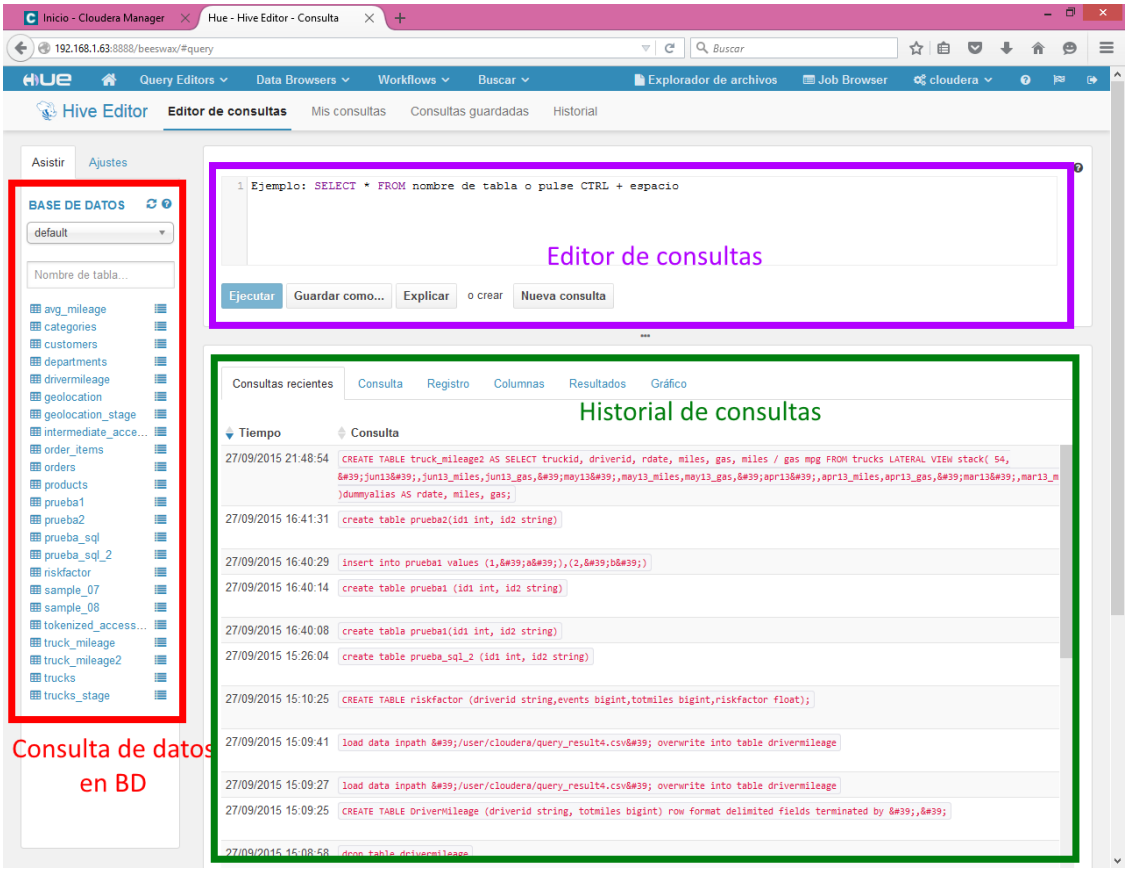


Figura 38. En *Hue* 3.7 (pre – instalado en *Cloudera*) el editor *Hive* es más completo presentando en una misma página el propio editor de consultas, navegación por la información de la base de datos y un historial de consultas realizadas

En cuanto a *Pig*, las dos interfaces pueden parecer iguales pero hemos encontrado que *Hue* 3.7 no presenta la opción de añadir argumentos a la ejecución, algo que puede ser necesario en ocasiones. Por otro lado, en cualquiera de las dos se puede acceder a un conjunto de plantillas con sentencias *Pig* básicas.

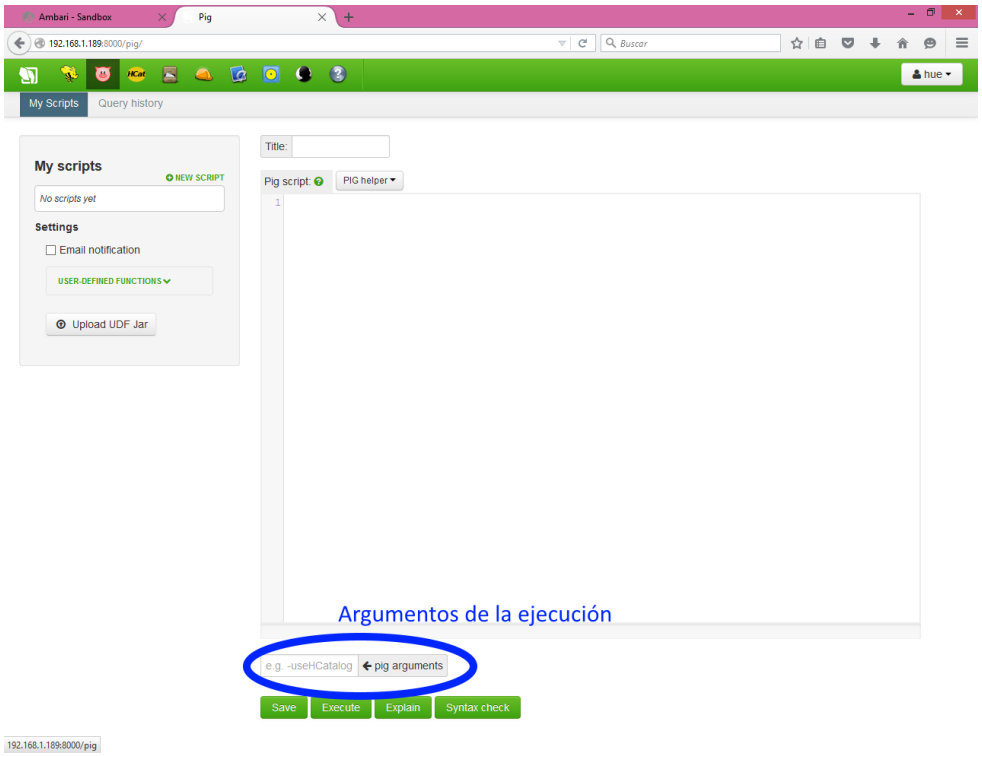


Figura 39. Editor *Pig* en *Hue* 2.6

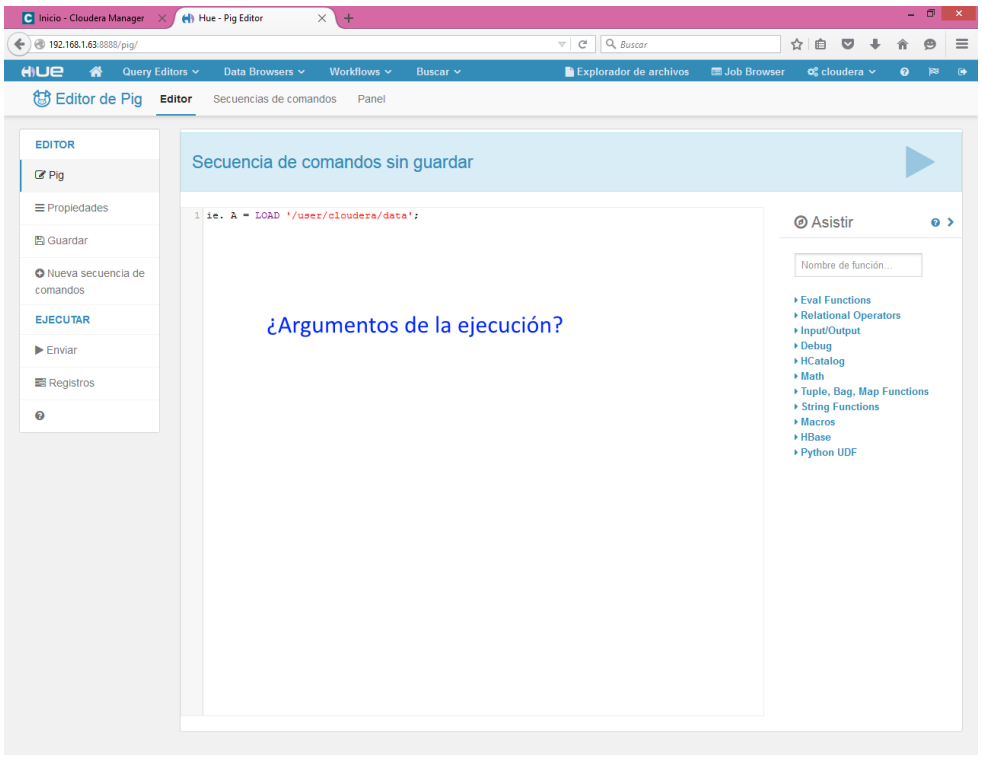


Figura 40. Editor *Pig* en *Hue* 3.7. No presenta la opción de indicar argumentos para la ejecución

Por último, en cuanto a *Oozie*, es más completa la interfaz de *Hue* 3.7, como se puede comprobar en la Figura 41. *Oozie* es el planificador de tareas y flujos que combinan acciones de distintos tipos (*Hive*, *Pig*, etc.). Consideramos que es más completa porque permite añadir acciones de más tipos, algo que en principio podría parecer restrictivo para escoger *Cloudera* como distribución *Hadoop*. Pero esto no va a ser un problema para nosotros ya se puede insta-

lar *Hue* 3.8 en *Hortonworks* y por lo tanto en cuanto a esta interfaz *Cloudera* y *Hortonworks* son iguales.

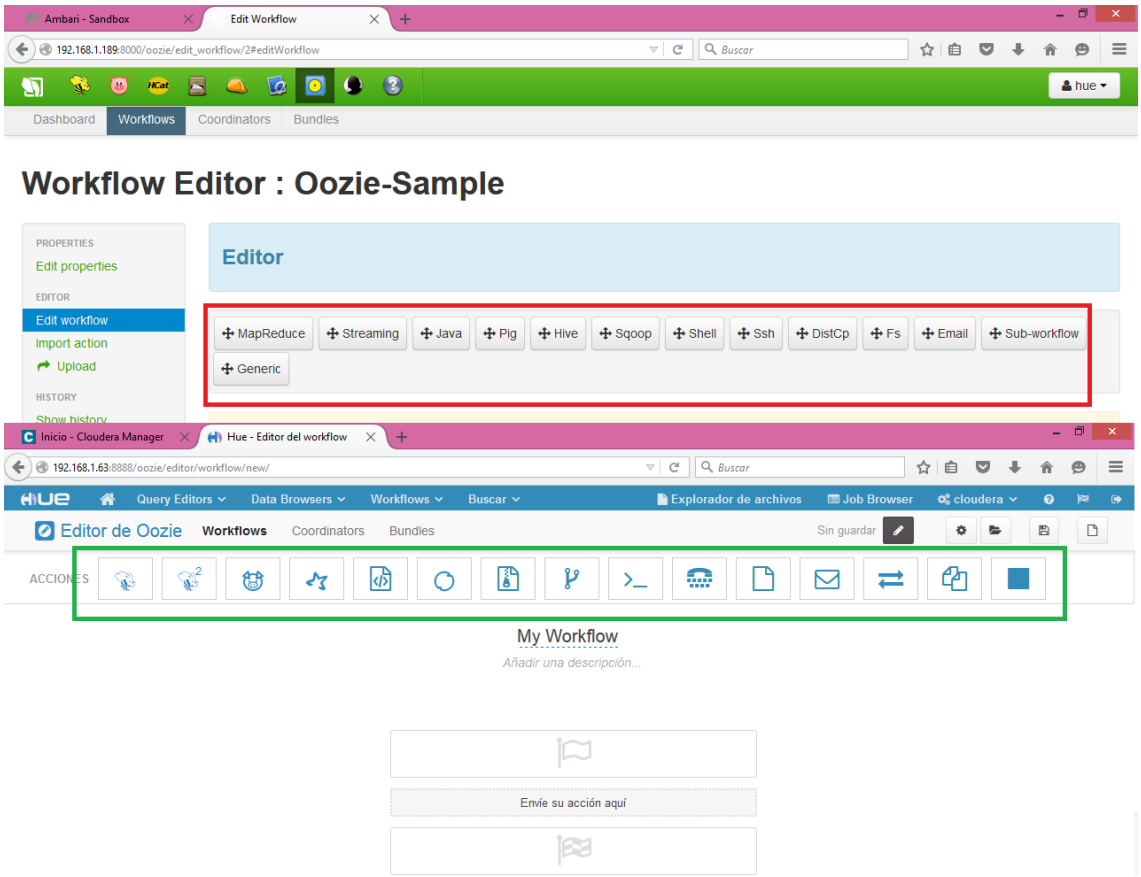


Figura 41. Comparativa entre las páginas de creación de flujos de trabajo *Oozie* en *Hue* 2.6 (superior) y *Hue* 3.7 (inferior)

Aunque no debería ser un punto importante a tener en cuenta, porque no debe ser un obstáculo, existen versiones en español de las interfaces *Cloudera Manager* y *Hue* 3.8 las cuales se presentan al usuario si este tiene seleccionado español como idioma preferido en el navegador *web* que utiliza para acceder a ellas.

Visto todo esto, poco tenemos para comparar ambas infraestructuras y decidirnos por una de ellas. En cuando a su manejo práctico son iguales, usan los mismos paquetes *software* para manejar *Hadoop* y la interfaz por medio de la cual esto se realiza es la misma (tras haber instalado manualmente *Hue* 3.8 en *Hortonworks*). A nivel de monitorización de servicios, estado del *cluster* y demás, parece más completa *Cloudera*, pero esto es un apartado que no nos preocupa en exceso. Todo esto nos inclina a usar *Hortonworks*, ya que en distintos sitios *web*, como [94], se comenta que para comenzar a usar *Hadoop* es la mejor de las alternativas. Esto, junto con el hecho de que las pruebas realizadas como cambiar configuraciones por defecto de los servicios o de la infraestructura, diseñar un *cluster* para computación en paralelo añadiendo nuevos nodos o añadir funcionalidades externas han sido exitosas sin mucha dificultad, se ha optado por escoger esta alternativa frente a seguir analizando *Cloudera*. Además, a vistas de las pruebas realizadas, parece que la infraestructura *Hortonworks* es menos estricta en cuanto a la memoria RAM necesaria para ejecutarse sin problemas, a pesar de que, según las guías de instalación, se requiere la misma cantidad de RAM en ambas distribuciones.

2.2.3.2. *Arquitectura Hadoop*

En base a la distribución de *Hadoop/MapReduce* que acabamos de seleccionar, en este apartado se explica la arquitectura que presenta. En concreto, hemos optado por escoger la versión más reciente, ***Hortonworks Data Plataform*** **(HDP) 2.3**, lanzada en Julio de 2015. En la Figura 42 se puede observar una evolución de las distintas versiones de HDP así como las distintas versiones de las tecnologías integradas con *Hadoop*.

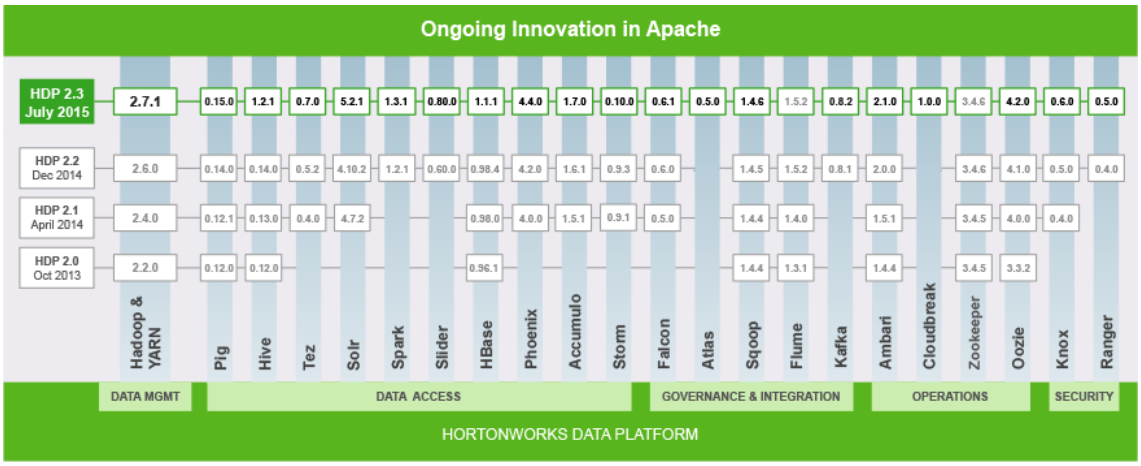


Figura 42. Evolución de HDP y sus tecnologías integradas. Fuente: [95]

Apache Hadoop [96]–[98] es un marco de trabajo de código abierto que permite el almacenamiento distribuido y procesamiento de grandes cantidad de datos a lo largo de un *cluster* de máquinas. Este *framework* se compone de cuatro módulos principales:

- *Hadoop Common*: librerías y utilidades necesarias para otros módulos *Hadoop*.
- *Hadoop Distributed File System* (*HDFS*): sistema de ficheros distribuido donde almacenar los datos, proporcionando un gran ancho de banda agregado a lo largo del *cluster*. Se analizará más adelante.
- *Hadoop YARN*: plataforma de gestión de recursos responsable de gestionar el procesamiento de recursos en el *cluster* y usarlos para programar las aplicaciones de usuario.
- *Hadoop MapReduce*. Modelo de programación usado para el procesado de los datos.

El elemento fundamental de *Hadoop* es la implementación del modelo *MapReduce*. ***MapReduce*** [99] es un modelo de programación y su implementación asociada para el procesamiento y generación de grandes cantidades de datos. Para ello, el usuario debe especificar dos funciones, denominadas *Map* y *Reduce* con el objetivo de generar un conjunto de salida de parejas clave/valor a partir de otro conjunto distinto de entrada. La función *Map* recibe una pareja de entrada y produce un conjunto intermedio de parejas clave/valor. La librería *MapReduce* agrupa juntos todos los valores intermedios que tengan asociada la misma clave intermedia. Estos resultados se pasan a la función *Reduce*. Esta recibe una clave intermedia y un conjunto de valores que cumplen esta clave, a partir de los cuales realiza el procesamiento que haya definido el usuario para quedarse con un conjunto más pequeño de valores, pudiendo incluso ser un conjunto vacío si el procesamiento de usuario no selecciona ninguno de los valores disponibles. Todo el sistema de ejecución tras las funciones de usuario se encargará de particionar los datos, programar la ejecución a lo largo de las máquinas disponibles, gestionar fallos o la comunicación entre máquinas. De esta forma, no hace falta ser un experto en programación paralela y distribuida para implementarla.

De forma esquemática, el funcionamiento de *MapReduce* se suele representar de la siguiente forma:

Map: (k1, v1) → list (k2, v2)

Reduce: (k2, list(v2)) → list(v2)

Es decir, la función *Map* convierte un conjunto de parejas clave/valor en una lista de parejas clave/valor. La función *reduce* toma una clave k2 y el conjunto de valores que tiene asociados para generar otra lista de valores v2. Por ejemplo, supongamos que tenemos un programa que cuenta el número de veces que cada palabra aparece en un texto [100]. Siguiendo el esquema de la Figura 43, podemos tener dos ficheros de entrada con dos líneas cada uno (paso 1) que representan el total del texto. Cada línea de cada fichero se puede pasar a una instancia de función *Map* distinta (paso 2). Esta función estará diseñada para, por cada palabra, generar una pareja clave/valor, donde la clave es la propia palabra y el valor "1", representando una ocurrencia (paso 3). Después, las librerías de *MapReduce* se encargarán de ordenar todos los resultados para "juntar" todas las parejas con la misma clave (paso 4). Cada uno de estos conjuntos pasará a una instancia de función *Reduce* donde, en este ejemplo, se contará el número de elementos del conjunto, sumando todos los valores de las parejas, para dar lugar al total de cada una de las palabras (pasos 5 y 6).

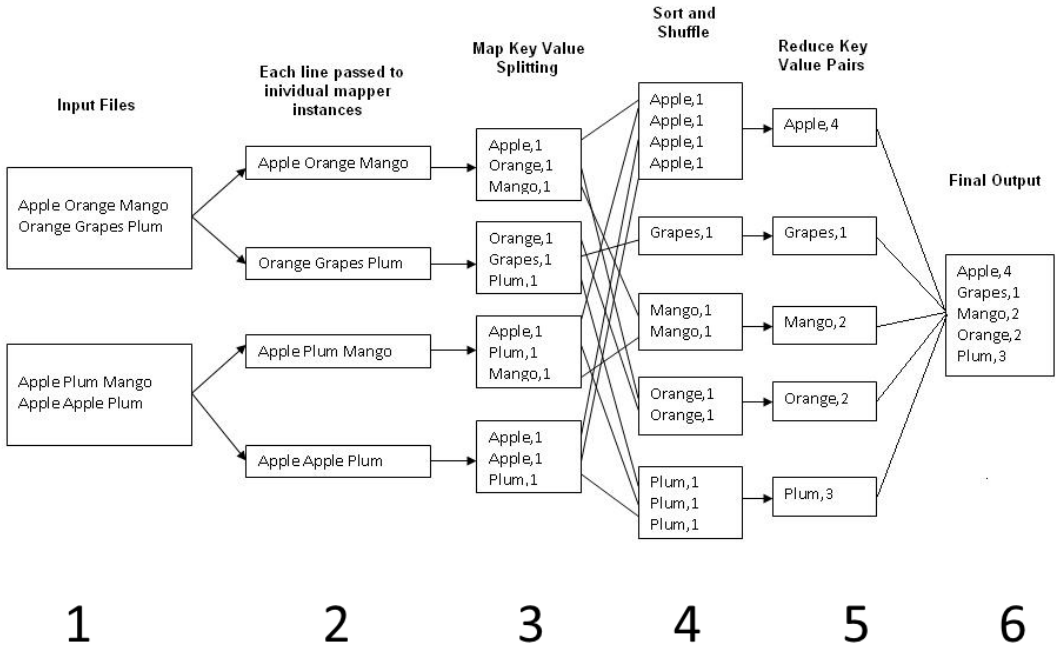


Figura 43. Ejemplo *MapReduce*: cuenta palabras. Fuente: [100]

El plan de ejecución genérico se presenta en la Figura 44. Cuando se lanza un trabajo *MapReduce*, tienen lugar la siguiente **secuencia de acciones** [99]:

1. El programa comienza cuando las librerías *MapReduce* dividen los datos de entrada a procesar en M bloques, típicamente entre 16 y 64 MB. Cada uno de estos bloques será procesado por una instancia de la función *Map*. Estas instancias se distribuyen por las máquinas disponibles para su procesado en paralelo, donde una misma máquina puede ejecutar varias instancias.

2. Una de las copias del programa es especial, es el maestro (*master*). El resto son trabajadores (*workers*) a los que el maestro asigna el trabajo que deben realizar. El maestro

monitoriza a los trabajadores y reparte las tareas *Map* entre aquellos que están en espera o con capacidad para procesar. De igual forma procederá con las tareas *Reduce*.

3. Cuando un trabajador recibe su orden del maestro lee el contenido de los datos de entrada que le corresponden. Con ellos, aplica la función *Map* definida por el usuario, lo cual generará un conjunto de parejas clave/valor intermedias que, en primer lugar, se almacenarán en memoria.

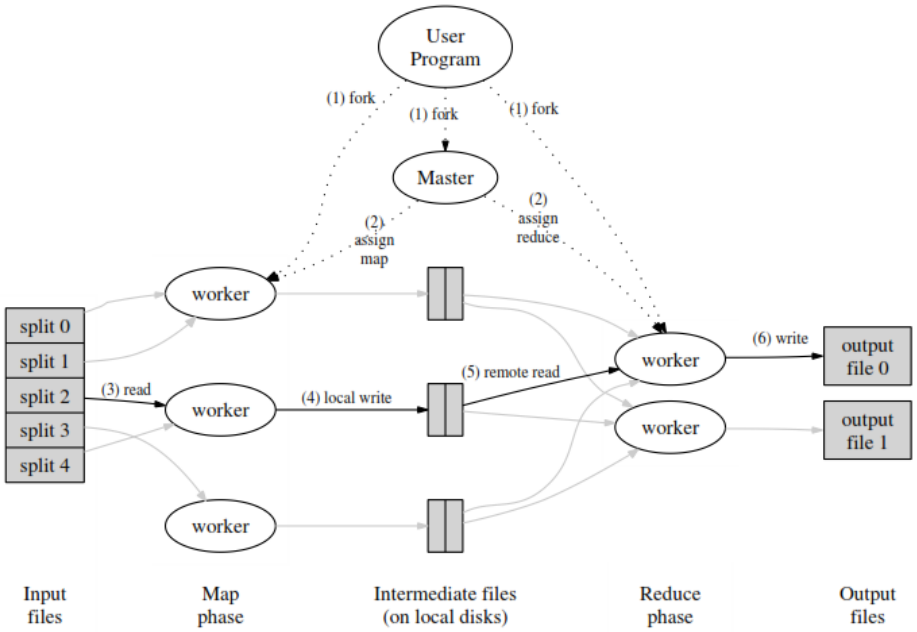


Figura 44. Plan de ejecución de un trabajo *MapReduce*. Fuente: [99]

4. Periódicamente, estos resultados intermedios se irán almacenando en el disco local a la vez que son particionados en R bloques. Además, el trabajador deberá notificar al maestro la localización en su disco local de estos bloques.

5. Sabiendo de esta localización, el maestro informa de ella a los trabajadores de la función *Reduce* para que, usando llamadas a procedimientos remotos, puedan leer los datos resultados de las funciones *Map*. A continuación, ordenan los datos agrupando las parejas con la misma clave.

6. Cada conjunto de parejas clave/valor con la misma clave se pasa a una función *Reduce* en la que se implementa la operación definida por el usuario. Los resultados correspondientes se escriben en el sistema de ficheros distribuido. La implementación de *MapReduce* es tal que se genera un fichero de salida por cada una de las funciones *Reduce*, ya que se considera que, por lo general, no hace falta combinar estos resultados en un único fichero. Si esto último es necesario, se suele hacer pasar estos resultados por una nueva función *MapReduce*.

En todo este procedimiento hay que distinguir dos conceptos: el sistema de ficheros distribuido sobre el que se lanza el trabajo y el sistema de ficheros local de cada máquina que conforma el *cluster*. En el sistema de ficheros distribuido se aloja el fichero de entrada y se escriben los resultados de la operación. En el sistema de ficheros local de cada nodo se almacenan los resultados intermedios. Es por ello que, para funcionar, *MapReduce* necesita de un sistema

de ficheros distribuido [101]. En la versión original, este es el sistema de ficheros de *Google*, *Google File System* (GFS)[1], pero *Hadoop* usa su propio sistema de ficheros distribuido, *HDFS*.

Un punto importante para la ejecución de un trabajo *MapReduce* es la **gestión de los fallos** [99] en los nodos del *cluster*, fallos que provocarán el fallo de elementos trabajadores o maestros. Para realizar dicha gestión, el maestro comprobará periódicamente, mediante *pings*, la disponibilidad de cada nodo trabajador. Si no recibe durante un cierto tiempo respuesta de uno de ellos, lo marcará como fallido y no lo tendrá en cuenta a la hora de repartir trabajo. Si el trabajador fallido había finalizado una tarea *Map*, esta se marcará como no iniciada, ya que sus resultados son inaccesibles al estar almacenados en el disco local de una máquina no disponible, y, por tanto, deberá ser re – ejecutada en otro nodo. Más inconvenientes trae el fallo del nodo maestro, ya que es el punto más sensible de la infraestructura. Para evitar la paralización total del entorno si falla este nodo, se suelen hacer copias periódicas de él para ponerlas inmediatamente en marcha si falla el maestro principal.

En el contexto concreto de *Apache Hadoop/MapReduce* [102], al nodo maestro se le denomina ***JobTracker***. El resto de nodos esclavos o trabajadores se denominan ***TaskTrackers***. El *JobTracker* es responsable de la gestión de recursos (gestionar los *TaskTrackers*), gestión del consumo de recursos y gestión del ciclo de ejecución de trabajos (planificación de tareas, seguimiento de progresos, proporcionar tolerancia a fallos, etc.). Por otro lado, cada *TaskTracker* tiene responsabilidades más simples: ejecutar las órdenes del *JobTracker* y proporcionarle información periódica sobre su estado. Un ejemplo de estructura de nodos puede apreciarse en la Figura 45.

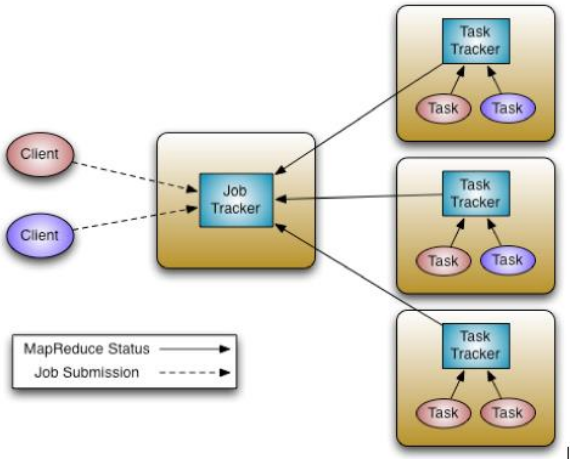


Figura 45. Arquitectura *MapReduce*: *JobTracker* & *TaskTracker*. Fuente: [102]

Entre las principales características [101], [103] de *Hadoop/MapReduce*, se podrían citar:

- Simplicidad: los trabajos *MapReduce* se pueden escribir en el lenguaje que al programador le parezca más adecuado: *Java*, *C++, Python*, etc.
- Escalabilidad: *MapReduce* puede procesar *petabytes* de datos, almacenados en *HDFS*.
- Rapidez: al realizar un procesado en paralelo, se pueden ejecutar tareas, que de forma tradicional llevarían días, en horas o minutos.

[1] La versión de *MapReduce* de *Google* no está disponible al público, es por ello que aun siendo un desarrollo suyo, la implementación más conocida es *Apache Hadoop/MapReduce*, ya que está si es de uso público.

- Recuperación: *MapReduce* tiene en cuenta posibles fallos que puedan ocurrir. Si una máquina con una copia de los datos no está disponible, se podrá encontrar esta información en otras máquinas.

- Mínimo movimiento de los datos: a la hora de realizar tareas, no se mueven los datos en el sistema de ficheros distribuido *HDFS* a los procesos, sino que son los procesos los que se mueven allá donde estén los datos, lo que minimiza el tránsito de datos por la red, aumentando la rapidez de procesamiento.

- Flexibilidad: *MapReduce* no depende de un modelo de datos o esquema concretos.

- Independiente del almacenamiento, solo requiere un sistema de ficheros distribuido, pero sin especificar uno en concreto.

- Alta escalabilidad, se soportan *clusters* compuestos por miles de nodos. Por ejemplo, *Yahoo!* implementó un *cluster Hadoop* con más de 4000 nodos en 2008 [104].

Aun así, hay una serie de puntos débiles [101]:

- No existe un lenguaje de alto nivel propio de *MapReduce*. A veces, codificar según qué trabajos en los lenguajes posibles puede ser bastante costoso.

- No tiene un esquema o índice.

- Cada operación que se quiera realizar debe poder describirse mediante un único trabajo *MapReduce*, algo que no siempre es posible.

- Baja eficiencia.

- Es una tecnología bastante reciente (2004) comparada con ciertas alternativas.

Anteriormente, se han comentado numerosos problemas asociados a *MapReduce*, situaciones en las que no podía utilizarse porque los trabajos no se podían adaptar a la estructura del modelo. Desde la versión 2.0 de *Hadoop* se incorpora dentro del entorno de trabajo ***YARN*** (*Yet Another Resource Negotiator*). *YARN* es una tecnología de gestión de *clusters* que aporta soporte para cargas de trabajo que no sea de tipo *MapReduce*, es decir, habilita a una infraestructura *Hadoop* a implementar tareas que no sigan el esquema *MapReduce*, lo cual extiende el número de aplicaciones que se pueden ejecutar, ya que, entre otros, permite el procesamiento en tiempo real.

Adoptando *YARN* aparece una nueva arquitectura maestro – esclavo, como la mostrada en la Figura 46 [102]. El nodo maestro se denomina ***ResourceManager*** y cada nodo esclavo constituye un ***NodeManager***. En esta infraestructura aparece un tercer elemento: por cada aplicación lanzada se ejecuta un demonio ***ApplicationMaster***[2].

La idea es tal que, el total de responsabilidades del *ResourceManager* y un *ApplicationMaster* es equivalente a las tareas que realizaba el *JobTracker* en *MapReduce*. Por lo demás, las responsabilidades de cada elemento son:

[2] En la práctica nos podemos encontrar *ApplicationMaster* que gestionan un conjunto de aplicaciones, por ejemplo, un *ApplicationMaster* para *Pig* o *Hive* que gestiona un conjunto de trabajos *MapReduce*

- El *ResourceManager* se encarga de la gestión de recursos. Para ello cuenta con un planificador (*Scheduler*) que se encargará de decidir cómo se debe realizar el reparto de recursos entre aplicaciones en ejecución teniendo en cuenta las características actuales de cada *NodeManager* para decidir a cuál asignar una tarea o si esta debe esperar a que se liberen recursos.

- El *NodeManager* de cada nodo esclavo es responsable de lanzar los contenedores donde se ejecutan las aplicaciones, monitorizar el uso que estas hacen de los recursos del nodo e informar periódicamente al *ResourceManager* de todo ello.

- Un *ApplicationMaster* no forma directamente parte de la estructura maestro – esclavo. Se define como una entidad o marco de trabajo que se encarga, por una parte de negociar, con el *Scheduler* del *ResourcerManager,* el contenedor de recursos donde se va a ejecutar su aplicación y por otra, de trabajar con los *NodeManager's* correspondientes para realizar un seguimiento de la aplicación lanzada y monitorizar su progreso.

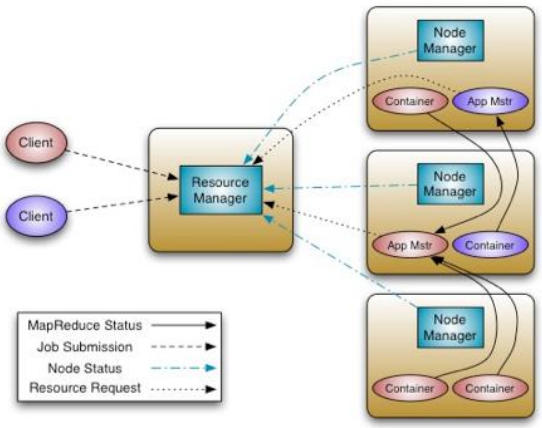


Figura 46. Arquitectura *Hadoop YARN: ResourceManager* y *NodeManager*. Fuente: [102]

La gran diferencia entre *MapReduce* y *YARN* está en el *ApplicationMaster* que proporciona dos grandes características a la nueva infraestructura *Hadoop*:

- Escalabilidad: proporcionando a cada aplicación su propio *ApplicationMaster* se puede conseguir implementar un *cluster* mucho más grande (en pruebas se ha conseguido hasta un *cluster* de 10000 nodos) sin pérdida de eficiencia. Además, como cada *ApplicationMaster* está particularizado a cada aplicación, ninguno de ellos provocará un cuello de botella.

- Se consigue generalizar el sistema, ya que al mover toda la definición de la aplicación al *ApplicationMaster* se da soporte no solo a modelos como *MapReduce* si no también cualquier otro como MPI o Procesado Gráfico (Ver Sección 2.2.1).

Pero, **¿cómo funciona esta nueva infraestructura?** Podemos seguir el diagrama de la Figura 47 [105]:

1. El cliente envía la aplicación a ejecutar con todas las especificaciones necesarias para su ejecución.

2. El *ResourceManager* negocia un contenedor donde iniciar el *ApplicationMaster* de esta aplicación y lo lanza en un determinado nodo esclavo.

3. Una vez lanzado, el *ApplicationMaster* se registra ante el *ResourceManager*.

4. El *ApplicationManager* negocia con el *Scheduler* del *ResourceManager* los recursos que se van a proporcionar a la aplicación.

5. El *Scheduler* le puede proporcionar recursos repartidos por varios nodos, incluso sin ser en el mismo nodo esclavo en que se ejecuta el *ApplicationManager*, y se lanza en ellos los contenedores de la aplicación.

6. Dentro de este conjunto de contenedores, la aplicación correspondiente se ejecuta, proporcionando la información necesaria a su *ApplicationMaster* para poder ser monitorizada.

7. A su vez, existe una comunicación directa entre el *ApplicationMaster* y el cliente para que este pueda ver también el estado de su aplicación.

8. Una vez que la aplicación se ha completado, el *ApplicationMaster* se desregistra del *ResourceManager* y se apaga, liberando los recursos y contenedores consumidos.

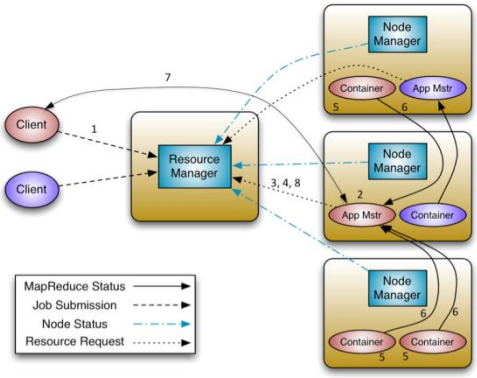


Figura 47. Flujo de trabajo del lanzamiento de una aplicación en *Hadoop YARN*. Fuente: [105]

Estos son los tres conceptos importantes a tener en cuenta al realizar este trabajo de *Big-Data*, pero, como se ha comentado, *Apache Hadoop* realmente gana valor cuando se usa junto con otras tantas tecnologías que facilitan aún más su uso. Estas tecnologías se pasan a enumerar a continuación. Principalmente hay que tener en cuenta que aunque se ha hablado de que una de las características de *MapReduce* es su sencillez de programación, ya que se puede escoger el lenguaje de programación que se quiera para diseñar las funciones *Map* y *Reduce*, en la práctica para el programador puede ser complicado abstraer cómo dividir una funcionalidad final que se pretende conseguir en una secuencia de parejas de funciones *Map* – *Reduce* que cumplan las especificaciones del modelo. Es por ello, que la mayor parte de las tecnologías que aquí se van a presentar tienen por objetivo ofrecer otras herramientas en las que se puede expresar una tarea a realizar, para que después y de forma transparente al usuario, la infraestructura *Hadoop* y la propia tecnología lo conviertan a un trabajo *MapReduce*. Así, ni tan siquiera el programador debe molestarse en definir las funciones *Map* o *Reduce*, la propia tecnología lo hace por él. Solo tiene que saber lo que esa tecnología le permite hacer, lo cual estará limitado por las capacidades de *MapReduce* y, en este caso además, *YARN*.

Debido a que vamos a utilizar la infraestructura de *Hortonworks*, nos centramos en las tecnologías que esta compañía proporciona integradas junto a *Hadoop*.

Hortonworks divide su estructura en cinco bloques, como se puede apreciar en la Figura 48, unos bloques que vamos a ir comentando poco a poco. Si uno aprecia el índice de esta sección, puede ver que aparece un sexto bloque denominado “Visualización de Resultados”, el cual no

está presente en esta figura. Esto es debido a que, no solo *Hortonworks* sino la mayor parte de distribuciones de *Hadoop*, no incluyen herramientas de visualización de los resultados generados mediante *Hadoop*. Si queremos generar gráficos con los análisis realizados deberemos integrar otras herramientas. En este caso se ha optado por integrar *ElasticSearch* y *Kibana*. Pero recordar que a pesar de estar aquí incluidas, no suelen formar parte de ninguna distribución *Hadoop*.

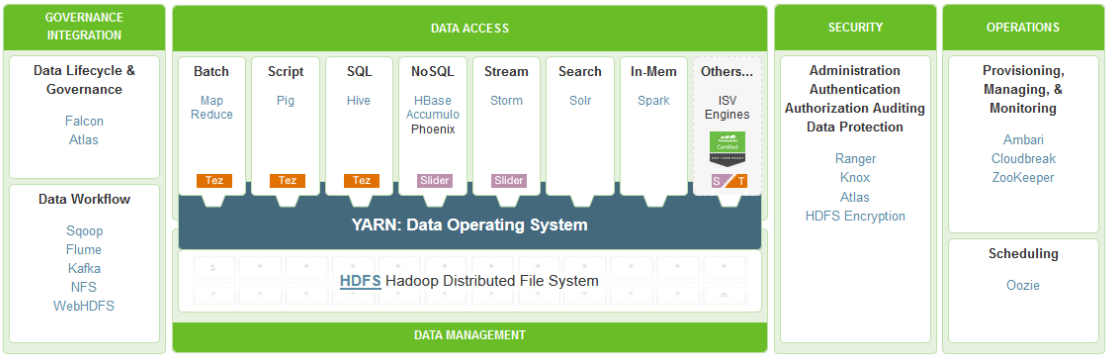


Figura 48. Arquitectura *Hadoop Hortonworks Data Platform* (HDP). Fuente: [106]

Las tecnologías de **gestión de datos** se van a encargar de almacenar y procesar enormes cantidades de datos [106]. Distinguimos dos tecnologías:

- ***Apache Hadoop YARN***, ya comentada.
- ***HDFS***. Ver sección 2.2.3.2.1.

Los módulos de **acceso a los datos** permiten interactuar con los datos de una gran cantidad de formas, desde procesamiento por lotes hasta en tiempo real. ***Apache Hive*** es la tecnología de acceso a los datos más ampliamente adoptada, aunque hay muchos motores especializados. Por otro lado, ***Apache Pig*** ofrece capacidades de *scripting*, ***Apache Storm*** ofrece procesamiento en tiempo real, ***Apache HBase*** ofrece almacenamiento *NoSQL* columnar y ***Apache Accumulo*** ofrece un control de acceso a nivel celular. Todos estos motores pueden trabajar a través de un conjunto de datos y recursos gracias a *YARN* y motores intermedios como *Apache Tez* para acceso interactivo y *Apache Slider* para aplicaciones de larga duración. *YARN* también proporciona métodos de acceso a los datos como ***Apache Solr*** para búsquedas y marcos de programación como ***Cascading***. Esta es la descripción que *Hortonworks* [106] hace de los módulos de acceso a los datos. Son un conjunto de tecnologías que, de formas muy distintas, permiten manipular los datos para realizar distintos análisis. Dependiendo de las necesidades de este análisis se podrá optar por una u otra herramienta. No son excluyentes, dependiendo de situaciones una será más apta que otra o se podrá usar una combinación de ellas. Lo importante es que permiten lanzar trabajos *MapReduce* (incluso no *MapReduce* gracias a *YARN*) sin necesidad de que el usuario divida su aplicación siguiendo este modelo. Será la propia herramienta la que se encargue de "traducir" el programa de usuario en el correspondiente lenguaje a un trabajo *MapReduce* para, después, *YARN* encargarse de ejecutarlo de forma distribuida a lo largo del *cluster*.

- ***Apache Accumulo*** [107], [108] proporciona almacenamiento de parejas clave/valor distribuido y ordenado con control de acceso basado en celdas, pero no es una base de datos relacional completa. Consiste en una gran tabla de almacenamiento de datos de baja latencia. Está basado en *Google BigTable* y se ejecuta sobre *HDFS* y *YARN*,

que proporciona aplicaciones de visualización y análisis de acceso predictible a los datos en *Accumulo*.

- ***Apache Hive***. Ver Sección 2.2.3.2.2.
- ***Apache Pig***. Ver sección 2.2.3.2.3.
- ***Apache Spark***. Ver sección 2.2.3.2.4.
- ***MapReduce***  . Explicado anteriormente.
- ***Apache*** **HBase**  [109], [110]. Sistema de almacenamiento de datos *NoSQL* orientado a columnas que proporciona acceso aleatorio de lectura/escritura en tiempo real.
- ***Apache Kafka***  [111], [112]. Sistema de colas de mensajes rápido y escalable. Se suele usar en lugar de los sistemas tradicionales debido a su alto rendimiento, replicación y tolerancia a fallos. Trabaja conjuntamente con *Apache Storm*, *Apache HBase* y *Apache Spark* para realizar análisis en tiempo real y manejar datos en *streaming*.
- ***Apache Slider***  [113], [114]. Entorno de trabajo para desplegar aplicaciones de acceso a los datos de larga duración en *Hadoop*. Para ello, aprovecha las capacidades de gestión de recursos de *YARN*, de tal forma que permite gestionar el ciclo de vida de las aplicaciones o escalarlas, incluso cuando están en ejecución. Además, permite crear y ejecutar distintas versiones de las aplicaciones. Cada instancia puede configurarse de forma distinta con su ciclo de vida de operación gestionado manualmente. *Apache HBase*, *Apache Accumulo* y *Apache Storm* utilizan *Apache Slider* como motor de ejecución.
- ***Apache Storm***  [87], [115]. Sistema de computación distribuido en tiempo real que proporciona un rápido procesamiento de grandes corrientes de datos añadiendo un procesamiento fiable de datos en tiempo real a las capacidades de *Apache Hadoop*. Es poderoso en escenarios que requieren análisis en tiempo real, aprendizaje automático y monitorización continua de operaciones.
- ***Apache Mahout***. Ver Sección 2.2.3.2.5.
- ***Apache Solr***  [116], [117]. Plataforma de código abierto que permite la búsqueda de datos almacenados en *Hadoop HDFS*. Proporciona búsquedas avanzadas de texto completo e indexación en casi tiempo real.
- ***Apache Tez***. Ver Sección 2.2.3.2.6.

El conjunto de tecnologías que se encuadran en la categoría de **Gobierno de datos e integración** permitirán cargar datos rápida y fácilmente así como gestionarlos siguiendo ciertas políticas.

- ***Apache Atlas***  [118], [119]. Conjunto de servicios de gobierno diseñados para intercambiar metadatos con otras herramientas y procesos tanto dentro como fuera de *Hadoop*.

- ***Apache Falcon*** [120], [121]. Entorno de gestión de datos que permite simplificar la gestión del ciclo de vida de los datos en *Hadoop*. Permite gestionar el movimiento de los datos, el procesado de tuberías, recuperación de desastres y flujos de trabajo de datos.

- ***Apache Flume*** [122], [123]. Permite agregar eficientemente y mover grandes cantidades de datos de *log* desde múltiples fuentes.

- ***Apache Sqoop***. Ver sección 2.2.3.2.7.

Como en cualquier otro entorno, es necesario tener mecanismos de **seguridad**. Estos van a permitir cumplir requerimientos de Autenticación, Autorización, Contabilidad y Protección de datos. Esta seguridad se proporciona en cada capa de *Hadoop*, desde *HDFS* hasta *YARN* incluyendo *Hive* o cualquier otro componente de acceso a los datos.

- ***Apache Knox*** [124], [125]. Proporciona un único punto de autenticación y acceso a *Hadoop*. Tiene por objetivo simplificar la seguridad de *Hadoop* para usuarios que acceden a los datos del *cluster* y ejecutan trabajos, y para operadores que controlan el acceso al *cluster*.

- ***Apache Ranger*** [126], [127]. Proporciona una administración central de políticas de seguridad según requisitos de autorización, autenticación, auditoria y protección de datos.

Los módulos de **Operaciones** permiten provisionar, gestionar, monitorizar y operar sobre el *cluster Hadoop.*

- ***Apache Oozie***. Ver Sección 2.2.3.2.8.

- ***Apache Ambari***. Ver Sección 2.2.3.2.9.

- ***Hue***. Ver Sección 2.2.3.2.9.

- ***Apache ZooKeeper***. Ver Sección 2.2.3.2.10.

2.2.3.2.1. *HDFS*

Hadoop Distributed File System (*HDFS*) [98], [128] es un sistema de ficheros distribuido basado en *Java* que proporciona un almacenamiento de datos escalable y fiable. Fue diseñado para abarcar un gran número de servidores en un *cluster*. Entre sus características podemos destacar:

- Conocimiento del *rack* de servidores: a la hora de programar tareas se tiene en cuenta la ubicación física de los nodos.

- Mínimo movimiento de los datos: son los procesos los que se mueven a los datos y no al revés, ya que las tareas de procesamiento se llevan a cabo en el nodo donde están los datos, lo que reduce el tráfico en la red y proporciona un ancho de banda agregado muy alto.

- Proporciona utilidades de diagnóstico dinámico de la salud del sistema de ficheros y re-equilibrado de datos en los diferentes nodos.
- Requiere mínima intervención del operador.
- Permite almacenar cantidades enormes de datos a lo largo de cientos de máquinas.
- Proporciona acceso en tiempo real ya que ha sido diseñado teniendo en cuenta que será un sistema de ficheros en el que rara vez se escribirá algo, pero del que se leerá en numerosas ocasiones, de esta forma se ha conseguido un procesamiento de datos más eficiente.
- Para funcionar no requiere de un conjunto *hardware* específico o muy estricto.
- Proporciona un acceso de baja latencia.

La estructura de este sistema de ficheros también presenta una arquitectura maestro – esclavo, mostrada en la Figura 49. En este caso, al nodo maestro se le denomina ***NameNode*** y a los múltiples nodos esclavos, ***DataNode***.

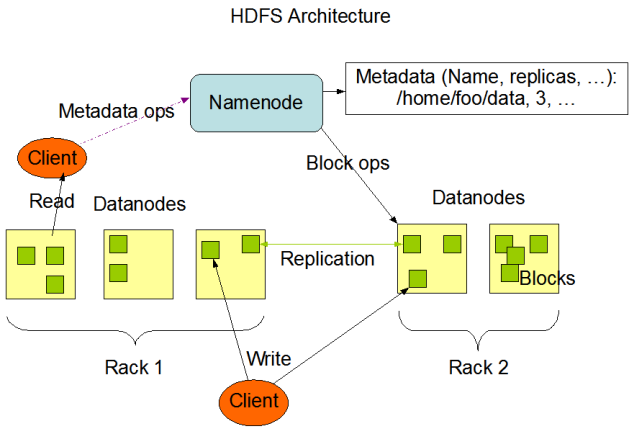


Figura 49. Arquitectura *HDFS*. Fuente: [128]

El nodo maestro *NameNode* es el responsable de gestionar los metadatos del *cluster*. En él, ficheros y directorios se representan por *i-nodos*, donde se almacena información sobre atributos, localización en *DataNodes*, etc., pero no almacena la propia información. Los nodos esclavos *DataNode* son los que almacenan los datos de directorios y ficheros y también se encargan de proporcionar estos datos cuando se solicitan a *HDFS*. De forma periódica informan al *NameNode* de la lista de datos que están guardando.

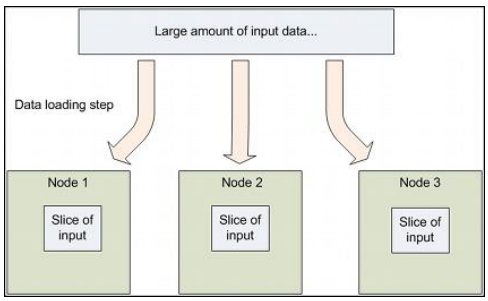


Figura 50. Particionado de datos en *HDFS*

En realidad no se almacenan datos como tal. Exactamente igual que en cualquier otro sistema de ficheros se almacenan bloques. En este caso, los bloques de datos son más grandes, de 128 MB[3]. Cuando se debe almacenar un nuevo fichero en *HDFS*, este se divide en bloques de 128 MB (Figura 50) y cada bloque se almacena independientemente replicado en múltiples *DataNode's* (en su sistema de ficheros local). El *NodeName* monitoriza activamente el número de réplicas de cada bloque. Si una de ellas se pierde por fallo en el nodo de datos, el *NameNode* crea otra réplica.

El *NameNode* se comunica con los *DataNode's* enviándoles instrucciones como respuesta a los "latidos" de estos. Instrucciones como replicar bloques a otros nodos, borrar replicas locales, apagar el nodo, etc.

A este sistema de ficheros se puede acceder de múltiples formas ya que existen varias interfaces para él [98]. En nuestro caso, al estar integrado dentro de una plataforma *Hadoop* plenamente configurada, no tendremos que preocuparnos por cómo configurar su acceso. Podremos manipular su contenido tanto desde línea de comandos, como vía *web* desde *Apache Ambari* y *Hue*, previamente comentadas. Manejarlo desde un navegador *web* es muy sencillo, ya que podemos ir navegando por los distintos directorios, borrar/añadir directorios o ficheros, modificar su nombre o contenido, ver su contenido, etc. Acceder desde línea de comandos puede ser ventajoso en ciertos casos, por ejemplo, para poder realizar tareas que requieran de ciertos privilegios que solo el usuario *hdfs* tiene. No mucha mayor complejidad tiene usar el terminal, ya que la mayor parte de comandos *UNIX* que se utilizan para gestionar el sistema de ficheros local se pueden usar en *HDFS*, utilizando una sentencia concreta.

2.2.3.2.2. *Apache Hive*

Apache Hive [129] consiste en un almacén de datos que facilita la consulta y la gestión de grandes cantidades de datos almacenados de forma distribuida. *Hive* proporciona un mecanismo para dar estructura a los datos almacenados y realizar consultas usando un lenguaje similar a *SQL* denominado ***HiveQL***. Estas consultas se convierten en trabajos *MapReduce* que se lanzan sobre la estructura de datos.

Este marco de trabajo proporciona [130]:

- Herramientas que facilitan *ETL* (Ver Capítulo 1) sobre los datos.
- Un mecanismo para imponer estructura en una variedad de formatos de datos.
- Acceso a ficheros almacenados en *HDFS* u otros sistemas de almacenamiento, como *Apache HBase*. Esto quiere decir que la información *Hive*, tratada como tablas con datos y atributos para los datos, se pueden guardar en distintos almacenamientos. En nuestro caso, en *HDFS* como ficheros. A su vez, estos ficheros pueden ser de distintos formatos, en nuestro caso serán ficheros ORC con contenido convenientemente etiquetado.
- Ejecución de consultas vía *MapReduce*.

[3] Los bloques de datos en *HDFS* son mayores que en cualquier sistema de ficheros genérico para poder minimizar el coste de las búsquedas. Si el bloque es lo suficientemente grande, el tiempo de transferencia puede ser significativamente más largo que el tiempo necesario para buscar el inicio del bloque [98].

- Permite añadir funciones *Map* y *Reduce* propias cuando estas no se pueden definir con la lógica *HiveQL*.

- *HiveQL* también se puede extender con funciones escalares customizadas (*UDF's*), agregaciones (*UDAF's*) y funciones de tabla (*UDTF's*).

El único punto débil está en que no soporta consultas en tiempo real. Principalmente está pensado para implementar trabajos por lotes sobre grandes cantidades de datos. Los puntos fuertes de *Hive* son la escalabilidad (a lo largo del *cluster Hadoop*), extensibilidad (marco *MapReduce* con *UDF*/*UDAF*/*UDTF*), tolerancia a fallos y sin acoplo con los formatos de entrada.

¿Cómo funciona *Hive*? [131]

Las tablas en *Hive* son similares a las tablas en cualquier base de datos relacional. Las bases de datos se componen de tablas. Los datos se acceden por medio de consultas *HiveQL*, incluso consultas para sobrescribir datos. Dentro de una base de datos en particular. Los datos en las tablas son serializados y cada tabla tiene su directorio en *HDFS* correspondiente, en la ruta `/apps/hive/warehouse` en el caso de *Hadoop Hortonworks Data Platform*. *Hive* soporta todas las primitivas de datos comunes como *BIGINT*, *BINARY*, *BOOLEAN*, *CHAR*, *DECIMAL*, *DOUBLE*, *FLOAT*, *INT*, *SMALLINT*, *STRING*, *TIMESTAMP* y *TINYINT*. A mayores, se pueden combinar tipos de datos primitivos para formar tipos de datos complejos, como estructuras, mapas o *arrays*, es decir, se puede configurar una tabla tal que una de sus columnas sea, por ejemplo, de tipo *array* de *strings*.

Dentro de *Hive* podemos distinguir dos **componentes**, *HCatalog* y *WebHCat* [130].

HCatalog [132], [133] es una capa de gestión de tablas y almacenamiento para *Hadoop*. Expone los metadatos de *Hive* a otras aplicaciones *Hadoop*, como *Pig* o *MapReduce*, de tal forma que sea más fácil leer los datos almacenados en el *cluster*. Presenta una "tabla de abstracción" que proporciona una vista relacional de los datos almacenados en *HDFS*, con los siguientes beneficios:

- Libra al usuario de tener que saber dónde están los datos o en qué formato están almacenados.

- Posibilita notificaciones sobre la disponibilidad de datos.

- Posibilita el uso de herramientas de limpieza de datos.

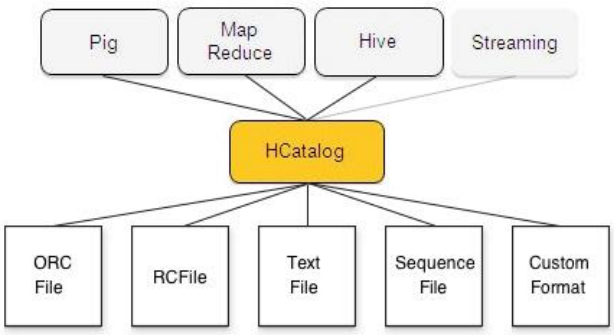


Figura 51. *HCatalog* sirve de nexo de unión entre los ficheros (en cualquier formato) y las herramientas *Hadoop*. Fuente: [132]

HCatalog soporta leer y escribir ficheros en cualquier formato para el cual se puede escribir un *SerDe* (*serializador – deserializador*) [134], es decir, una interfaz mediante la cual *Hive* pueda interpretar el contenido del fichero o crear contenido de acuerdo a un formato. Existe una serie de interfaces *SerDe* estandarizadas para formatos *ORC*, *RCFile*, texto, *JSON*, etc., pero se puede leer cualquier formato propio de usuario, si este especifica una pareja de interfaces de entrada/salida para interpretar los datos. *HCatalog* reúne la información extraída de estos ficheros en cualquier formato, para que pueda ser usada por otras herramientas (Figura 51).

Arquitectura de *HCatalog*. *HCatalog* se encuentra sobre la *metastore* de *Hive* e incorpora los *DDL* de Hive (*Hive Data Definition Lenguaje*, conjunto de sentencias *HiveQL* básicas). Para que otras herramientas puedan acceder a través de él a los datos, proporciona interfaces:

- *HCatLoader* y *HCatStorer* para cargar y escribir desde la herramienta *Pig*. Ambas herramientas aceptan una tabla almacenada en la infraestructura *Hive – HCatalog*, la primera para extraer y leer su contenido y la segunda para escribir contenido en ella, por lo general añadiendo sin sobrescribir lo ya existente.
- *HCatInputFormat* y *HCatOutputFormat* hacen lo propio para los trabajos *MapReduce*.
- Como *HCatalog* usa los metadatos de *Hive*, no existe interfaz para "comunicarlos", un trabajo *Hive* puede leer o escribir directamente los datos de *HCatalog,* sin necesidad de una interfaz para ello.

En cuanto al modelo de datos, se ha comentado previamente que *HCatalog* proporciona una vista relacional de los datos, en la que estos son almacenados en tablas y estas, a su vez, en bases de datos. Además, estas tablas se pueden dividir en una o más claves, es decir, para un determinado valor de clave (o conjunto de claves), existirá una partición que contenga todas las líneas de la tabla con esa clave (o conjunto de claves).

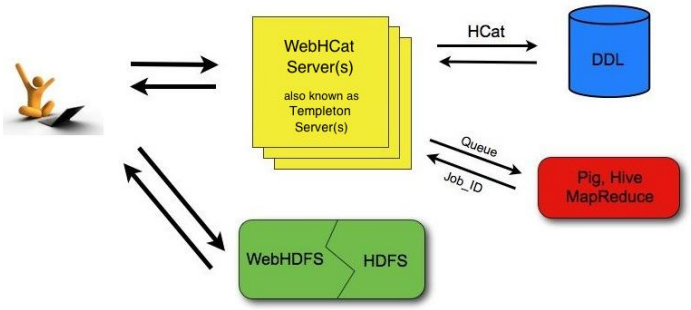


Figura 52. Flujo de trabajo típico usando *WebHCat*. Fuente: [135]

WebHCat (anteriormente *Templeton*) [135] proporciona un servicio que se puede usar para ejecutar trabajos *Hadoop MapReduce*, *Pig* o *Hive* a través de una interfaz *web* de tipo *REST*. El funcionamiento típico de esta interfaz se muestra en la Figura 52. El lado cliente realiza una petición para acceder a *Hadoop/MapReduce*, *YARN*, *Pig*, *Hive* o *HCatalog DDL*[4] desde la aplicación correspondiente. Los datos y el código usados por esta *API* se mantienen en *HDFS*. Los comandos *HCatalog DDL* se ejecutan directamente, pero los trabajos *MapReduce*, *Pig* o *Hive*

[4] *HCatalog DDL* consiste en aquel conjunto de sentencias *HiveQL* que no suponen ejecutar un trabajo *MapReduce*. Sentencias tales como aquellas que incluyen consultas `create/drop/alter table`, `create/drop/alter view`, `show/describe`, etc. La definición concreta de sentencias puede encontrarse en [136].

son encolados por los servidores *WebHCat*, de tal forma que se puede monitorizar su progreso o detenerlos cuando sea necesario. A su vez, el usuario debe especificar en qué punto de *HDFS* quiere guardar los resultados de los diversos trabajos.

En esencia es una *API REST* a través de la cual, mediante peticiones *HTTP* típicas (*GET*, *PUT*, *POST*, *DELETE*) se pueden gestionar los trabajos *MapReduce*, *Pig* y *Hive* lanzados, así como las sentencias *HCatalog DDL*. En nuestro trabajo no utilizaremos, al menos directamente, esta interfaz para gestionar los trabajos. Pero las interfaces que presenta *Hortonworks* (*Apache Ambari* y *Hue*) utilizan esta interfaz para lanzar algunos de los trabajos, por lo que en realidad y de forma transparente sí que la estamos usando.

A mayores, *Hive* presenta una interfaz gráfica de usuario, alternativa a usar *Hive* por línea de comandos, denominada ***Hive Web Interface*** (*HWI*) [137] en la que no incidiremos mucho más ya que haremos uso de las que proporcionan *Apache Ambari* y *Hue*.

En cuanto al lenguaje ***HiveQL***, es bastante similar a *SQL*, pero añade una gran lista de nuevas funcionalidades. En [138] se puede consultar el manual de este lenguaje. Aspectos interesantes que se pueden destacar son:

- Posibilidad de usar el formato *ORC* (*Optimized Row Columnar*) para almacenar de forma altamente eficiente los datos en *Hive*. Usando ficheros *ORC* se mejora el rendimiento cuando *Hive* lee, escribe y procesa datos. Existen otros formatos posibles, ya indicados en la Figura 51, pero este se considera uno de los más eficientes.

- Posibilidad de indicar cómo están limitados los datos (mediante comas, tabuladores, saltos de línea, etc.) en el fichero en el formato en que se guardan en *Hive* para, después, poder leerlos adecuadamente.

- Existencia de función adicionales que permiten añadir operaciones que en *SQL* no existen, como operaciones aritméticas (exponenciales, logaritmos, potencias), gran variedad de funciones para generar/convertir fechas en distintos formatos, funciones condicionales, operaciones estadísticas (correlaciones, covarianzas, percentiles), entre otros muchos.

- Y mucho más.

Todo ello hace que sea un lenguaje mucho más completo que *SQL*, obviamente sus objetivos son distintos. En [139] se puede encontrar una comparativa básica entre *SQL* y *HiveQL*; en ella se puede ver que las sentencias simples o elementales son iguales, la única diferencia está en toda la lista de funcionalidades nuevas que nos podemos encontrar en *HiveQL* para poder realizar análisis de datos.

2.2.3.2.3. *Apache Pig* 

Apache Pig [140], [141] es una plataforma que permite analizar grandes cantidades de datos. Consiste en un lenguaje de alto nivel, denominado *Pig Latin*, mediante el cual se definen programas de análisis de datos. La principal característica de los programas *Pig* es que son altamente susceptibles de ser paralelizados y por lo tanto ideales para implementar *pipelines* de operaciones *ETL* (*Extract – Transform – Load*) sobre los datos, búsquedas en datos sin manipular (*raw data*) o procesamiento iterativo de datos.

¿Cómo funciona *Pig*? [142], [143]

La infraestructura *Pig* consiste en un compilador que "traduce" los programas en *Pig Latin* en programas *MapReduce* que se pueden ejecutar a través de *YARN* accediendo a los datos necesarios almacenados en *HDFS*. De esta forma, se pueden crear complejos programas *MapReduce* pero a través de un lenguaje, *Pig Latin*, mucho más sencillo, facilitando la tarea al programador.

Pig se puede ejecutar de varios **modos** [143]:

- Modo local, en una única máquina y usando el sistema de ficheros de esta máquina. Los trabajos se ejecutan en una máquina virtual java, JVM.
- Modo local *Tez*, análogo al anterior pero utilizando como motor de ejecución *Apache Tez*.
- Modo *MapReduce*, el programa se ejecuta en el *cluster Hadoop* usando *HDFS* como sistema de ficheros. El programa se traduce a un trabajo *MapReduce*
- Modo *Tez*, como el anterior, pero usando *Apache Tez* como motor de ejecución.

De entre todos estos modos, *MapReduce* es el modo de ejecución por defecto, el único que no es necesario indicar y el que se utiliza al lanzar un programa *Pig* a través de las interfaces *Apache Ambari* y *Hue* en la *sandbox* de *Hortonworks*. Mediante ellas también se puede usar el modo *Tez* (además es recomendable), pero no lo modos locales (tampoco tendría mucho sentido configurar un *cluster* y luego lanzar los trabajos en máquinas individuales).

Luego, cada uno de estos modos de ejecución se pueden lanzar de forma interactiva o modo *batch* (por lotes), pero mediante las interfaces anteriormente citadas se usa la ejecución por lotes. En la Tabla 5 se puede apreciar la posibilidad de combinar las distintas formas de ejecución.

	Modo local	Modo local *Tez*	Modo *MapReduce*	Modo *Tez*
Modo Interactivo	Si	Experimental	Si	Si
Modo *Batch*	Si	Experimental	Si	Si

Tabla 5. Modos de ejecución de *Pig*

El usar uno u otro modo tiene sobre todo sentido si se ejecuta *Pig* desde línea de comandos, ya que a través de *Ambari* o *Hue* tiene una configuración predefinida que, se supone, es la más eficiente.

¿Cómo es el lenguaje *Pig Latin*?

La ventaja de *Apache Pig* es que nos permite escribir programas *MapReduce* de una forma mucho más sencilla que si debemos especificar las funciones *Map* y *Reduce* de nuestra aplicación, por ejemplo, en *Java* (que sería lo más coherente en *Hadoop* ya que esta es un entorno desarrollado en *Java*). *Pig Latin* es un lenguaje con un aspecto similar a *SQL* (aunque se pueden encontrar opiniones contrarias) pero que proporciona muchas más funcionalidades y caracterizado por [140], [143]:

- Ser fácil de programar. Permite generar complejos programas *MapReduce* mediante unas simples sentencias.
- Debido a la forma en que las aplicaciones se codifican, ofrece la oportunidad de optimizar su ejecución.

- Es un lenguaje extensible al que el usuario puede añadir sus propias funciones para realizar un procesamiento concreto.

- Normalmente las sentencias se ejecutan usando una ejecución multi – consulta (se procesan todas las líneas del programa a la vez).

Un programa en *Pig Latin* suele organizarse en tres etapas [143]:

- Primero, se cuenta con una sentencia *LOAD* mediante la cual se leen los datos del sistema de ficheros.

- A continuación, se procesan o transforman esos datos, según requiera el programa.

- Por último, se genera la salida del programa, bien mediante una sentencia *DUMP* para ver los resultados o una sentencia *STORE* para guardarlos.

La parte de la transformación de los datos es la que más alternativas puede tener. *Pig Latin* nos proporciona:

- Operadores aritméticos [144], como sumas, restas, operaciones booleanas, comparadores, etc.

- Operaciones relacionales [144], como filtros (seleccionar ciertos datos), agrupaciones (agrupar varios datos en una colección), uniones, iteradores (*foreach*), operadores para ordenar colecciones, etc.

- Funciones integradas [145]:

    - Funciones de evaluación como cálculo de máximos, mínimos, valor medio, contar ocurrencias, etc.

    - Funciones de carga/almacenamiento.

    - Funciones matemáticas como cálculo del valor absoluto, logaritmos, exponenciales, funciones trigonométricas, raíz cuadrada, etc.

    - Funciones para manipular cadenas *String*.

    - Funciones relacionadas con la fecha.

    - Se pueden invocar las funciones *UDF* definidas sobre *Hive*.

- Funciones definidas por el usuario *UDF's* [146]. Estas pueden ser implementadas en *Java* (lo más común y además plenamente soportado), *Jython*, *Python*, *JavaScript*, *Ruby* o *Groovy*. Una vez que se ha definido la función de usuario, se debe generar con ella un ejecutable (por ejemplo *JAR* para *Java*) y añadirlo a *Pig*. Entre ellas cabe destacar el conjunto de funciones denominadas *PiggyBank* [147], desarrolladas por usuarios *Pig* y puestas al alcance del resto, u otras colecciones [148] como *DataFu*. Por ejemplo, en *Hortonworks*, están disponibles tanto *PiggyBank* como *DataFu*, ampliando las posibilidades de *Pig Latin*.

- Además, se pueden embeber operaciones *Pig* dentro de programas *Java*, *Python*, *JavaScript* o *Groovy*, es decir, incluir sentencias *Pig* dentro de otro lenguaje por medio de una interfaz [149]. Por ejemplo, para embeber *Pig* dentro de *Java* se usa la interfaz *PigServer*.

2.2.3.2.4. *Apache Spark*

Apache Spark [86], [150], [151] es una plataforma de computación distribuida de código abierto que ofrece un alto rendimiento para procesamiento tanto iterativo como interactivo, facilitando el desarrollo de algoritmos de aprendizaje automático (*machine learning*) que requieren un acceso iterativo y rápido a los datos. Se **caracteriza** por:

- Ser hasta 100 veces más rápido que *Apache Hadoop* usando *MapReduce* en memoria, y 10 veces más rápido haciendo escrituras en disco.
- Es fácil de usar. *Spark* no ofrece ningún lenguaje nuevo. Los programas o aplicaciones se pueden escribir en lenguajes como *Java*, *Scala*, *Python* o *R* (usando librerías propias de *Spark*), después se compilan y se pasan al entorno *Spark* que se encargará de ejecutarlos.
- Se puede ejecutar tanto en modo *standalone*, como sobre otras infraestructuras como *Hadoop*, *Mesos* o incluso en la nube. Además, puede acceder a datos de múltiples fuentes como *HDFS* o *HBase*.
- Es un entorno bastante genérico. Está compuesto por **librerías** que engloban conceptos bastantes distintos (Figura 53):
    - *SQL* y *DataFrames* [152] dan soporte para manejar datos estructurados y consultas relacionales. Ofrece compatibilidad con *Hive.*
    - *MLlib* [153], librería integrada para implementar algoritmos de *machine learning*.
    - *GraphX* [154], nueva *API* de *Spark* para procesamiento gráfico.
    - *Spark Streaming* [155] para el procesamiento de datos en tiempo real.

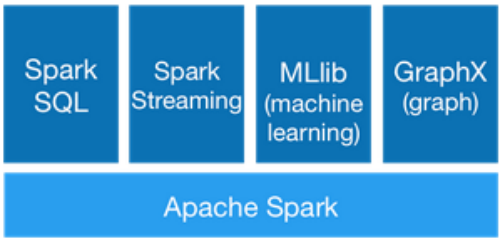


Figura 53. Librerías componentes de *Apache Spark*. Fuente: [86]

Por otro lado, existen diversas **formas de ejecutar** aplicaciones en *Spark* [155]:

- Localmente, en la propia máquina y solamente en ella.
- De forma distribuida a lo largo de un *cluster* [156]. Para ello se requiere de un gestor de *cluster*. Este puede ser:
    - *Spark Standalone* [157]. Este es un modo de despliegue proporcionado por *Spark* para poder ejecutar aplicaciones sin la necesidad de instalar cualquier otro entorno de gestión de *clusters*.
    - Otro gestor de *cluster*, como *Apache Hadoop YARN* o *Apache Mesos*. Si se tiene disponible una infraestructura de gestión de *cluster*, se puede hacer uso de ella

para desplegar *Spark*. La distribución *Hortonworks* empleada solo tiene soporte para *YARN*.

En el caso de optar por una ejecución distribuida, el diagrama de flujo genérico sería el mostrado en la Figura 54. Para gestionar las aplicaciones, *Spark* existe un programa principal, denominado *Driver Program* que se encarga de inicializar al *SparkContext* (encargado de coordinar un conjunto de aplicaciones *Spark* independientes). Este es el que se encarga de comunicarse con el gestor del *cluster* correspondiente (por ejemplo, *YARN*) para negociar los recursos que se van a ofrecer a las aplicaciones. Una vez que el gestor tiene a bien dar recursos a la aplicación, se reservan para ella unos ejecutores (*executor* en la Figura 54), los procesos en los que se llevará a cabo la computación y se almacenarán los datos de esa aplicación. Para ello, *SparkContext* deberá enviar el código a ejecutar junto con las tareas a realizar sobre él. En nuestro caso, como gestor de *cluster* tenemos *YARN*, así que como *Cluster Manager* tendríamos al *ResourceManager* junto con el *ApplicationMaster* asignado a la aplicación, como *Worker Nodes's*, los *NodeManager* y los procesos correrían en contenedores (*containers*).

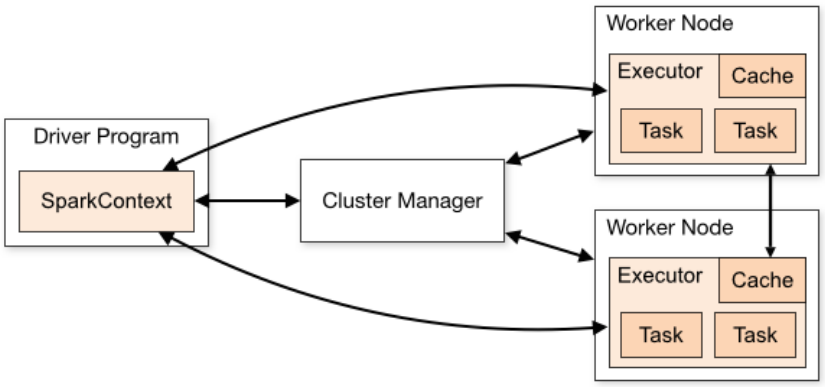


Figura 54. Ejecución genérica de un programa *Spark* en modo distribuido. Fuente: [157]

Aun dentro del modo de ejecución distribuido debemos distinguir otros dos **modos de despliegue**, al menos disponibles en *YARN* [158], que es el gestor de *clusters* que nos interesa:

- *Modo Cluster* (Figura 55): el *driver* de la aplicación *Spark* se ejecuta en el *Aplication-Master* asociado en un *host* del *cluster* escogido por *YARN*. Esto significa, que un mismo proceso ejecutándose en un contenedor, será el responsable de guiar la aplicación y de solicitar recursos para ella. Esto, además, implica que el cliente que lanza la aplicación no necesite estar ejecutándose durante todo el tiempo que dure la ejecución de la aplicación, ya que su "base" se ha trasladado a otro nodo del *cluster*.

- *Modo Cliente* (Figura 56): en este caso el *driver* de la aplicación *Spark* se ejecuta en la misma máquina desde la que se lanzó el trabajo. El *ApplicationMaster* asociado solo se encargará de solicitar un contenedor donde ejecutar la aplicación. Esto implica, que, en este caso, sí que se necesita que el cliente siga ejecutándose durante todo el tiempo de ejecución del trabajo, tal que si se detiene el cliente, se detendrá la aplicación.

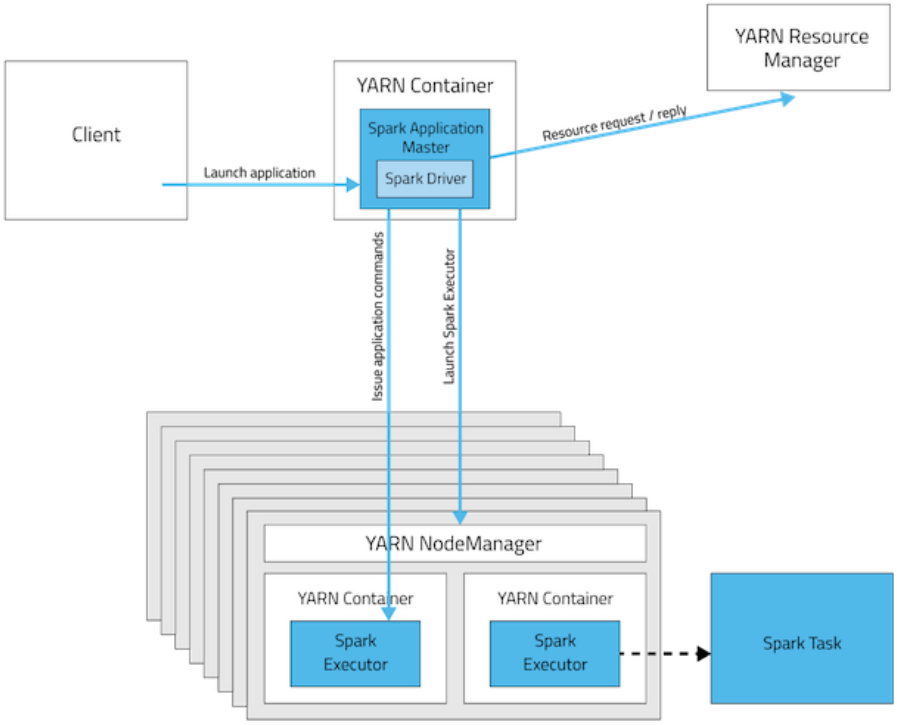


Figura 55. Diagrama de actividades en el modo de despliegue *YARN-Cluster*. Fuente: [158]

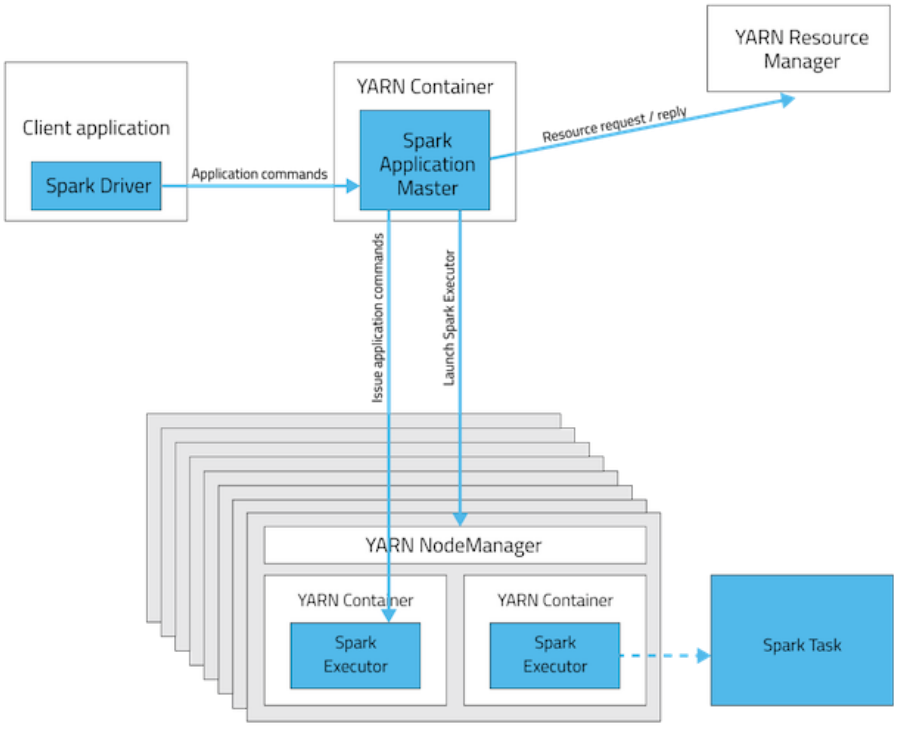


Figura 56. Diagrama de actividades en el modo de despliegue *YARN-Client*. Fuente: [158]

En la Tabla 6 se presenta un resumen de los distintos modos de despliegue de *Spark* así como su significado [158]. En la Tabla 7 se muestra un resumen comparativo entre las tres formas de despliegue distribuido a nivel de qué lleva a cabo cada acción [158].

Master - Modo de despliegue[5]	Significado
Local	*Spark* se ejecuta de forma local, con un solo hilo de trabajo (sin paralelismo)
Local [k]	*Spark* se ejecuta de forma local con K hilos de trabajo
Local [*]	*Spark* se ejecuta de forma local con tantos hilos de trabajo como núcleos tenga la máquina
Standalone	*Spark* se ejecuta de forma *standalone* en una máquina que tenga disponible el gestor de *clusters Spark Standalone*
YARN – cluster	*Spark* se ejecuta a lo largo de un *cluster YARN* en modo *cluster*
YARN – client	*Spark* se ejecuta a lo largo de un *cluster YARN* en modo cliente

Tabla 6. Modos de despliegue de *Spark*. Fuente: [159]

Modo	*YARN – Client*	*YARN – Cluster*	*Spark Standalone*
¿Dónde se ejecuta el driver de *Spark*?	En el cliente que lanza la aplicación	En el *ApplicationMaster* asociado a la aplicación lanzada	En el cliente que lanza la aplicación
¿Quién realiza la petición de recursos?	El *ApplicationMaster*	El *ApplicationMaster*	El cliente que lanza la aplicación
¿Dónde se inician los procesos a ejecutar?	En los nodos *NodeManager* disponibles	En los nodos *NodeManager* disponibles	En los nodos *Spark Worker* disponibles

Tabla 7. Resumen modos ejecución *Spark*. Fuente: [159]

Anteriormente, se presentó *Spark* como una alternativa a *Hadoop*, y así es entendido. Se dice que el futuro de *BigData* está marcado por *Spark* y ya no tanto por *Hadoop* [160], principalmente debido a la mayor rapidez que tiene *Spark* para realizar el mismo trabajo que *Hadoop*. Teniendo esto presente, empresas distribuidoras de *Hadoop* como *Hortonworks* o *Cloudera*, incluyen en sus distribuciones *Apache Spark* como un componente integrado sobre la infraestructura *Hadoop*. De esta forma, se pueden ejecutar trabajos *MapReduce*, *Hive* o *Pig* sobre *Hadoop* pero también se tiene la posibilidad de ejecutar trabajos sobre *Spark* y este a su vez sobre *YARN*, y poder aprovechar las ventajas o características de ambos entornos y utilizar el que sea más conveniente en cada caso concreto, evitando la creación de un *cluster* específicamente dedicado a *Spark*.

Pero, **¿qué es lo que diferencia a *Spark* de *Hadoop* para trabajar más rápido?** La rapidez de ejecución de trabajo no es la única diferencia entre ambos, pero sí la más destacable [161]. Ambos son marcos de trabajo que permiten un procesamiento distribuido de grandes cantidades de datos, pero presentan diferencias en su ejecución.

Hadoop permite un procesamiento distribuido usando un procesamiento por lotes mediante *MapReduce*. *Spark* mejora el algoritmo *MapReduce*, el cual no está optimizado para algoritmos iterativos o análisis de datos iterativos que realizan operaciones sobre el mismo conjunto de datos. Ambos ejecutan el algoritmo *MapReduce* pero con algunas diferencias, siendo la

[5] Siendo precisos, para lanzar una aplicación se indica el tipo de *Master* (*local*, *local* [k], *local* [*], una máquina con *Spark Standalone* o *YARN*). En caso de seleccionar *YARN*, se debe especificar el tipo de modo de despliegue: *cluster* o cliente (por defecto). Con *Master* se selecciona el tipo de ejecución de la aplicación, si se ejecuta en modo *YARN* a mayores tiene para elegir el modo de despliegue, aunque a efectos teóricos se pueden considerar todos modos de despliegue, pero a la hora de configurarlo son conceptos distintos.

principal que *Hadoop* está basado en escritura en disco y *Spark* en escritura en memoria. Esto puede apreciarse en la Figura 57. Después de cada etapa (entendida como una operación *Map* o *Reduce* si se concatenan varios trabajos *MapReduce*), los resultados se escriben en disco, mientras que con *Spark*, resultados intermedios se almacenan en memoria (salvo que no haya espacio, entonces se almacenan en disco) y solo se almacena en disco el resultado final. Esto explica por qué *Spark* puede llegar a ejecutar trabajos de procesamiento por lotes entre 10 y 100 veces más rápido que *Hadoop*.

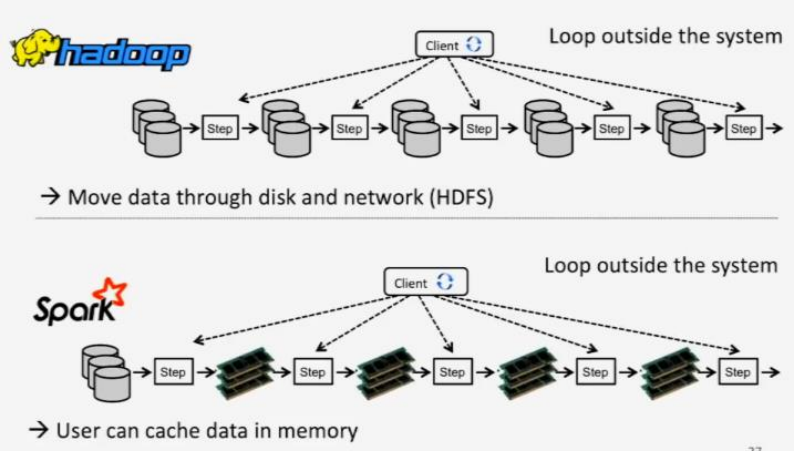


Figura 57. Comparativa *Hadoop/MapReduce* vs *Spark* a nivel de ejecución. Fuente: [162]

Fuera de la rapidez de procesamiento, *Spark* es mucho más fácil de gestionar. Integrando *Spark* en *Hadoop* se pueden realizar procesamientos en tiempo real, por lotes o algoritmos de *machine learning*. Además, con *Spark* se pueden controlar diferentes tipos de cargas.

Por otro lado, *Spark* permite realizar computaciones iterativas (que necesitan pasar muchas veces por los mismos datos). Pero si solo se va a pasar una vez por trabajo por lo datos, *Hadoop MapReduce* es la alternativa ideal.

A nivel de coste, para el funcionamiento adecuado de *Spark* se requiere una mayor cantidad de memoria. De esta forma, si queremos resolver un problema *BigData* con dimensiones reales, será mejor opción *Hadoop* ya que el tamaño del almacenamiento en disco es mayor y más barato que utilizar dispositivos de memoria.

En cuanto a la tolerancia a fallos, si mientras un proceso está en ejecución se produce un fallo, en *Hadoop* se continúa desde el punto de fallo, mientras que con *Spark* es necesario volver a empezar desde el principio.

Por último, *Hadoop* es hoy en día más seguro que *Spark*.

Otro aspecto a tener en cuenta son las capacidades del programador. Realizar *BigData* por medio de *Hadoop* implica utilizar *Hive* y *Pig*, principalmente, y por tanto conocer lenguajes de aspecto similar a *SQL*. Utilizar *Spark* implica saber programar en *Java*, *Scala* o *Python*. Las habilidades del programador también pueden ser un elemento que incline la balanza hacia una opción u otra.

Uno de los puntos de *Apache Spark* que nos pueden interesar, para desarrollar nuestro trabajo, es la librería *MLlib* [163], dedicada a la implementación de algoritmos de *machine learning*. En la Tabla 8 se resumen los algoritmos implementados así como su definición.

Categoría	Algoritmo		Descripción
Herramientas estadísticas básicas	**Correlación**		Cálculo de la correlación entre dos conjuntos de datos
	Muestreo estratificado (*stratified sampling)*		Método utilizado para obtener muestras de una población
	Testeo de hipótesis (*hypothesis testing*)		Permite determinar si un resultado es estadísticamente significativo, es decir, si se produjo o no por casualidad
	Generación aleatoria de datos (*random data generation*)		Método útil para algoritmos aleatorios, creación de prototipos y pruebas de rendimiento
Clasificación (identificar a qué categoría pertenece un dato) y regresión (estimar relaciones entre variables)	**SVMs**		Partiendo de un conjunto de muestras de entrenamiento, se genera una serie de clases en base a las cuales se quiere clasificar las nuevas muestras reales
	Regresión logística (*logistic regression*)		Permite predecir el resultado de una variable categórica, es decir, una variable que puede adoptar un número limitado de valores.
	Regresión: **mínimos cuadrados** (*least squares*), Lasso y regresión cresta (*ridge regression*)		Uso del algoritmo de mínimos cuadrados para resolver problemas estadísticos
	Regresión lineal en *streaming*		Permite implementar un algoritmo de regresión sobre los datos cuando estos se reciben en tiempo real
	Árboles de decisión (*decision trees*)		Se basan en tomar sucesivamente decisiones dividiendo el espacio de decisión de forma binaria
	Conjuntos de árboles de decisión (*ensembles of decision trees*)	***Random forest***	Entrena sobre distintos conjunto de árboles de decisión por separado de forma paralela. La combinación de las predicciones de cada árbol reduce la varianza de las predicciones
		Árboles de gradiente potenciado (*gradient-boosted trees*)	Itera sobre la misma secuencia de árboles de decisión de forma continua. Los resultados de decisión de una iteración se usan para corregir errores anteriores
	***Bayes* Ingenuo** (*Naive Bayes*)		Clasificador probabilístico fundamentado en el teorema de *Bayes*. Calcula la función de distribución condicional de cada característica dada una categoría de clasificación, la aplica al teorema de *Bayes* para calcular la distribución de probabilidad condicionada de la categoría dada la observación para usarla en predicción.
	Regresión isotónica (*isotonic regression*)		Dados un conjunto finito de números representando la respuesta observada se buscan los valores que minimizan la función matemática que caracteriza al modelo

Filtrado Colaborativo	**Mínimo cuadrados alternados** (ALS)	Método de filtrado colaborativo usado en *MLlib* para implementar sistemas de recomendación.
***Clustering*: Agrupación de elementos siguiendo criterios**	**K – Medias** (K – Means)	Agrupa puntos de datos en una de k posibles categorías atendiendo a la valor medio de la categoría (calculado a partir de los valores de los puntos que ya contiene) y el valor del punto bajo análisis
	Mezcla gaussiana (*Gaussian Mixture)*	Representa una distribución compuesta en la que cada punto se extrae de una de k sub-distribuciones gaussianas, cada una con su propia probabilidad
	Agrupación por método de las potencias (*power iteration clustering*, *PIC*)	Permite agrupar vértices de un grafo en función de similitudes, como las propiedades de enlace del grafo. Utiliza el método de la potencia para calcula un *eigenvector* de la matriz de datos para usarlo para agrupar vértices.
	Asignación *Dirichlet* latente (*Latent Dirichlet Allocation*)	Permite inferir temas de una colección de documentos de textos
	K – Medias *streaming* (*streaming k-means*)	Versión del algoritmo k – medias para procesar datos en *streaming*
Reducción de dimensionalidad: Proceso de reducir el número de variables en consideración. Se usa para extraer características latentes de características en bruto o ruidosas o comprimir datos.	**Descomposición en valores singulares** (*singular value descomposition*, *SVD*)	Aplicación de la factorización SVD con fines estadísticos
	Análisis de componente principal (*principal component analysis*, *PCA*)	Método estadístico para hallar las causas de la variabilidad de un conjunto de datos y ordenarlas por importancia.
Extracción de características y transformación	***TF-IDF***	Método de vectorización ampliamente utilizado en minería de texto para reflejar la importancia de un término en un documento
	Word2Vec	Calcula la representación vectorial distribuida de palabras
	StandScaler	Estandariza características escalando a varianza unidad y/o eliminando la media utilizando las estadísticas de muestras de un conjunto de datos de entrenamiento. Es una etapa típica de pre – procesamiento
	Normalizer	Escala muestras individuales

	Feature Selection	Permite seleccionar las características más relevantes para usar en la construcción de modelos
	ElementwiseProduct	Multiplica cada columna del conjunto de datos por un escalar
	PCA	Permite proyectar vectores a un espacio de pocas dimensiones
Minería de patrones frecuentes	***FP – growth***	Permite extraer elementos frecuentes
	Reglas de asociación (*association rules*)	Implementa un algoritmo de generación de reglas en paralelo
	PrefixSpan	Algoritmo de minería de patrones secuencial

Tabla 8. Algoritmos de *Machine Learning* implementado en la librería *MLlib* de *Spark*

2.2.3.2.5. *Apache Mahout*

Apache Mahout [28], [164] es una librería de algoritmos de *machine learning* implementados sobre *Hadoop* y que hacen uso del paradigma *MapReduce* para su resolución. *Machine Learning* es lo que se conoce en español como aprendizaje automático. Se define como una disciplina de inteligencia artificial centrada en permitir que las máquinas puedan aprender sin necesidad de haber sido explícitamente programadas. Es un procedimiento muy común para mejorar el rendimiento futuro en base a comportamientos pasados.

Los algoritmos implementados se pueden ejecutar en modo local o en modo distribuido a lo largo del *cluster Hadoop*. Cada uno de estos algoritmos se puede ejecutar invocándolos a través del comando *Mahout* desde línea de comando. Una desventaja de esta librería es que todavía no tiene desarrollada interfaz gráfica, por lo que hay que manejarla desde consola. Podemos incluir cualquiera de estos algoritmos en un programa más completo haciendo uso de las interfaces que proporciona esta librería, por ejemplo, escribimos un programa en *Java* y accedemos a la librería *Mahout* para usar un determinado algoritmo como parte del programa, incluyendo la llamada y el método correspondientes a partir de unos datos de entrada para obtener unos datos de salida que seguimos utilizando en el resto de nuestro programa *Java*.

Apache Mahout no se utiliza solo sobre *Hadoop*, es por ello que no todos los algoritmos que estén implementados puedan ser compatibles con *Hadoop*. En [165] se puede consultar una lista de los algoritmos implementados así como sobre qué infraestructura se pueden ejecutar por ser compatibles. En la Tabla 9 se muestra un resumen de los algoritmos *Mahout* soportados en *Hadoop*.

Categoría	Algoritmo	Descripción
Filtrado colaborativo	**Filtrado colaborativo basado en elementos distribuidos** (*Distributed Item-based Collaborative Filtering*)	Estima la preferencia del usuario por un elemento buscando sus preferencias por elementos similares
	Filtrado colaborativo usando una factorización matricial paralela (*Collaborative Filtering Using a Parallel Matrix Factorization*)	A partir de una matriz de elementos que el usuario aún no ha visto, predice qué elementos el usuario podría preferir

Clustering	***Clustering Canopy*** (*Canopy Clustering*)	Para pre – procesar los datos antes de usar el algoritmo de las k – medias o *clustering* jerárquico
	Clustering* de procesos de *Dirichlet (*Dirichlet Process Clustering*)	Modelado de mezcla *bayesiano*
	K – Medias *Fuzzy* (*Fuzzy K-Means*)	Categoriza los elementos correspondientes en grupos, denominados *cluster*, "*soft*" donde cada punto puede pertenecer a más de un *cluster*.
	***Clustering* jerárquico** (*Hierarchical Clustering*)	Crea una jerarquía de *clusters* utilizando una aproximación de abajo a arriba o de arriba a abajo
	***Clustering* K – Medias** (*K-Means Clustering*)	Permite dividir n observaciones en k grupos o *clusters*, donde cada observación pertenecerá al *cluster* (solo uno) cuyo valor medio se aproxime más al de la observación
	Asignación *Dirichlet* latente (*Latent Dirichlet Allocation*)	Automáticamente y de forma conjunta agrupa palabras por temas y documentos por mezclas de temas
	***Clustering* de desplazamiento medio** (*Mean Shift Clustering*)	Permite buscar modos o *clusters* en un espacio bidimensional, donde el número de *clusters* es desconocido
	Clustering MinHash (*Minhash Clustering*)	Permite estimar rápidamente la similitud entre dos conjuntos de datos
	***Clustering* espectral** (*Spectral Clustering*)	Agrupa puntos usando *eigenvectores* de matrices obtenidas a partir de los datos
Clasificación	**Bayesiano** (*Bayesian*)	Usado para clasificar objetos en categorías binarias
	Random Forest (*Random Forests*)	Método de aprendizaje para la clasificación (y regresión) usando árboles de decisión
Minería de conjuntos de datos frecuentes	**Algoritmo de crecimiento paralelo FP** (*Parallel FP Growth Algorithm*)	Analiza elementos en un grupo e identifica que elementos típicamente aparecen juntos.
Reducción	**Descomposición en valores singulares** (*SVD*)	Aplicación de la factorización *SVD* con fines estadísticos
	Algoritmo *Lanczos*	Algoritmo iterativo que permite encontrar los *eigenvalores* y *eigenvectores* más útiles de un sistema lineal
	***SVD* Estocástico**	Algoritmo similar a *SVD* pero produciendo una descomposición *SVD* de rango reducido
	PCA	Permite hallar las causas de la variabilidad de un conjunto de datos y ordenarlas por importancia.
	Descomposición *QR*	Aplicación de la descomposición QR con fines estadísticos

Varios	***RowSimilarityJob***	Realiza un cálculo de la similitud de pares en documentos mediante *MapReduce*
	ConcatMatrices	No se ha encontrado información
	Collocations	Se define como *collocations* (colocaciones) a aquellas secuencias de palabras o términos cuya co-ocurrencia es más frecuente que lo esperado por azar
	TF-IDF	Permite la creación de vectores desde texto
	XML Parsing	Utilidades varias relacionadas con ficheros *XML*
	Email Archive Parsing	Utilidades varias relacionadas con ficheros de *eMail*

Tabla 9. Algoritmos de *machine learning* de *Mahout* soportados en *Hadoop*

2.2.3.2.6. *Apache Tez*

Apache Tez [166], [167] es un motor de ejecución extensible que permite construir aplicaciones para el procesamiento de datos tanto por lotes como interactivas, coordinadas por *Hadoop YARN*. *Apache Hive* y *Apache Pig* usan *Tez*.

Tez generaliza el paradigma *MapReduce*, mejorando su rapidez, expresando la computación mediante un gráfico de flujo de datos [168], [169]. Así se permite que mediante *Hadoop* se pueda hacer frente a otras cargas de trabajo, por ejemplo, algoritmos de *Machine Learning*. La principal diferencia entre ambos se puede apreciar comparando los diagramas de la Figura 58. Si se ejecuta un trabajo *MapReduce* (*Hive* o *Pig*) en *Tez*, en lugar de usar múltiples trabajos *MapReduce* concatenados para completar una tarea, se usa un modelo *MapReduceReduce MRR* consistente en una única etapa *Map* (con múltiples tareas *Map* en paralelo) seguida de varias etapas *Reduce*. Esto último se consigue llevando los datos de salida de una etapa *Reduce* de un procesador a otro sin escribir ningún resultado en *HDFS*. En el diagrama original de *MapReduce*, los resultados intermedios entre trabajos *MapReduce* deben escribirse en disco.

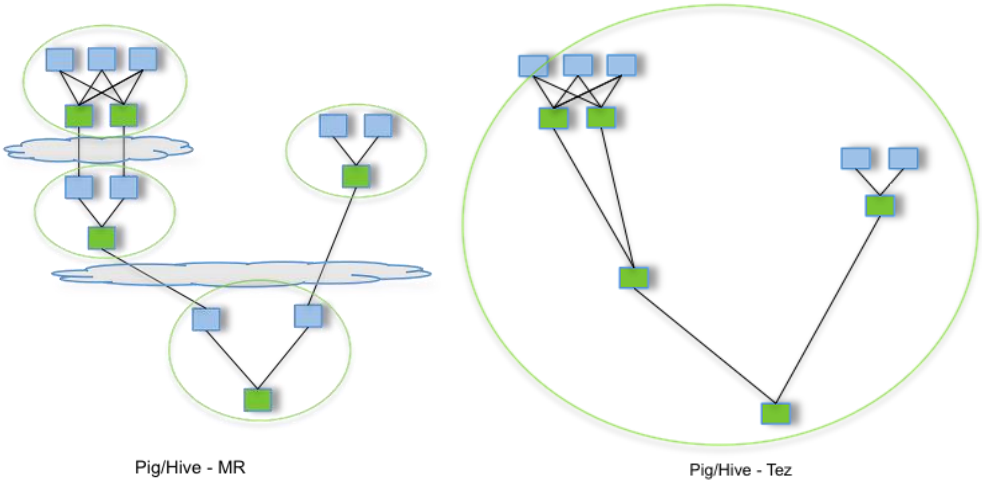


Figura 58. Comparativa ejecución trabajos *MapReduce* vs *Apache Tez*. Fuente: [168]

Apache Tez modela el procesamiento de datos mediante gráficos de flujos de datos. Los vértices de estos gráficos representan el procesado de los datos y los enlaces, el movimiento de los datos entre procesos. Estos gráficos tienen la forma de gráficos acíclicos directos (*DAG*).

El procesamiento comienza en el vértice raíz y continúa por los enlaces directos recorriendo todos los vértices hoja. Cuando todos los vértices del diagrama se han completado, finaliza el trabajo. Puede verse un diagrama de ejemplo en la Figura 59. En ella podemos volver a ver una comparativa entre *MapReduce* y *Tez* con un ejemplo concreto de programa en *Hive*. Este programa está compuesto por cuatro sentencias *Hive*, cada una de las cuales se vería como un vértice del diagrama de *Tez*.

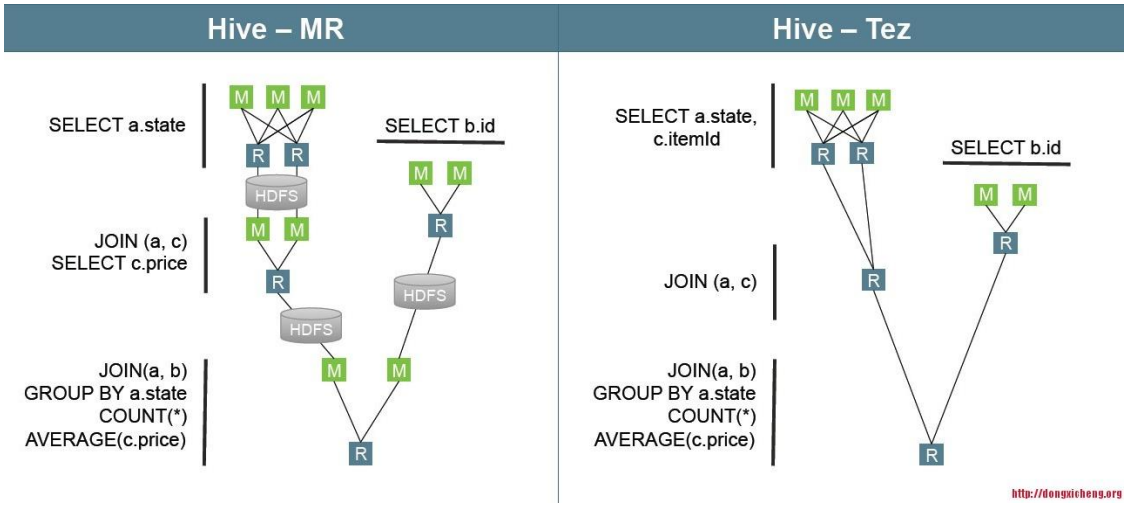


Figura 59. Ejemplo de diagrama DAG *Apache Tez – Hive*. Fuente: [170]

2.2.3.2.7. *Apache Sqoop*

Apache Sqoop [171], [172] es una herramienta diseñada para transferencia masiva de datos (en ambos sentidos) entre *Apache Hadoop* y estructuras de almacenamiento de datos como bases de datos relacionales, como *Teradata*, *Netezza*, *Oracle*, *MySQL*, *Postgres* o *HSQLDB* (Figura 60). Su funcionamiento está coordinado por *YARN* y entre sus capacidades se pueden encuadrar:

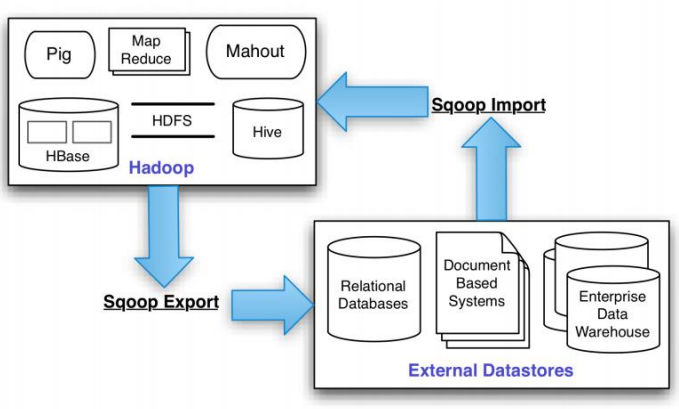


Figura 60. *Sqoop* permite transferir datos entre *Hadoop* y almacenamientos externos en ambos sentidos. Fuente: [173]

- Importación de conjuntos de datos secuenciales. De esta forma se satisface la necesidad creciente de importar/exportar datos hacia/desde *HDFS*.
- Posibilidad de una transferencia paralela de datos.

- Rápida realización de copias de datos.
- Balanceo de carga.

¿Cómo funciona *Sqoop*?

Sqoop proporciona un mecanismo de conectividad óptima con los sistemas externos [174]. Además, cuenta con una *API* que permite desarrollar nuevos conectores para ofrecer conectividad a nuevos sistemas, ya que por defecto solo vienen desarrollados conectores hacia los sistemas de almacenamiento de datos más populares.

Automatiza la mayor parte de su proceso, basado en la base de datos correspondiente para describir el esquema para los datos que se importan. Para importar y exportar datos utiliza *MapReduce*, siendo esta la clave de su funcionamiento en paralelo y tolerancia a fallos.

De entre todas sus utilidades, son dos las de mayor importancia [175]: las herramientas para importar y exportar datos. Con *Sqoop* se pueden importar datos desde una base de datos o central de datos (*mainframe*) hacia *HDFS*. La entrada al proceso sería la tabla de la base de datos o un conjunto de datos del mainframe. Como la operación de importación se realiza en paralelo con subconjuntos de los datos, la salida consistirá en múltiples ficheros que contienen la copia de la tabla o datos importados,

La **importación** se divide en dos etapas [176]: en primer lugar *Sqoop* analiza la base de datos para obtener los metadatos necesarios sobre los datos a importar, en segundo lugar implementa un conjunto de trabajos *MapReduce* con solo etapa *Map* para llevar los datos desde el almacenamiento externo a *Hadoop*, usando los metadatos adquiridos en la etapa previa (Figura 61). Además, existen opciones con las que se pueden importar directamente datos de una base de datos en *Hive*, *HBase* o *Accumulo*. En nuestro caso, parece interesante centrarnos en *Hive*. Con *Sqoop* podemos importar datos desde una base de datos externa a *Hive.* Especificando las opciones correspondientes, *Sqoop* generará una sentencia `CREATE` con la que creará la tabla y otra `LOAD DATA INPATH` para cargar los datos. En caso de que la tabla ya exista, se puede indicar que sobrescriba la tabla anterior sustituyendo su contenido.

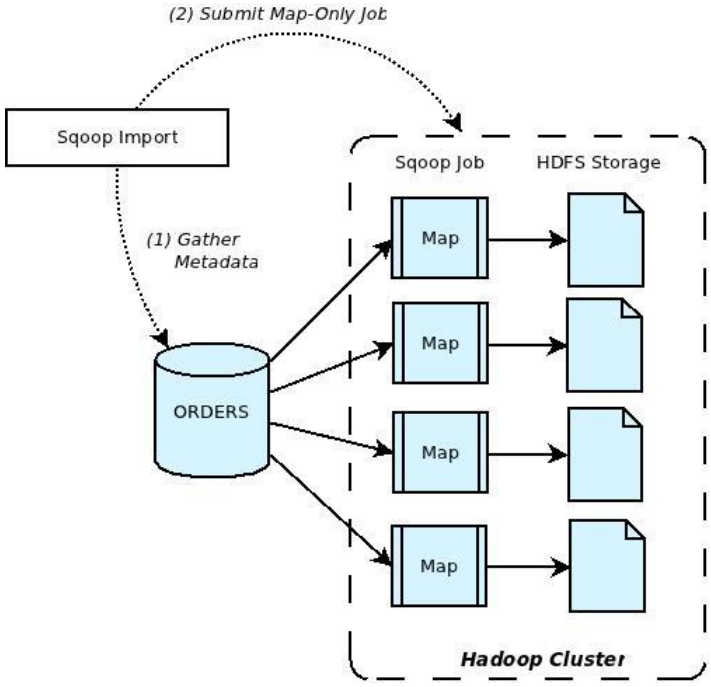


Figura 61. Proceso de importación en *Sqoop*. Fuente: [176]

Por otro lado, una vez manipulados los datos en *Hadoop*, estos se pueden exportar fuera de la infraestructura mediante el proceso de **exportación** de *Sqoop*. Este proceso lee los datos a exportar desde *HDFS* en paralelo, los convierte al formato correspondiente y los inserta como una nueva entrada en la tabla de la base de datos seleccionada por el usuario, transformando la orden de exportación en una consulta de tipo `INSERT`, es decir, no sobrescribe los datos sino que añade nuevas entradas conservando las ya existentes. Existe la opción de actualizar los datos de la tabla destino, especialmente pensado para aquellos casos en que existe una clave única en la tabla y un nuevo dato a añadir tiene que modificar los valores de esa clave, porque una inserción de un nuevo dato con misma clave única dará error. Para ello, usando la opción correspondiente e indicando la clave que se tomará para buscar la entrada de la tabla, se ejecutará una sentencia UPDATE para modificar su valor. Este proceso también consta de dos etapas [176]: en primer lugar se analiza la base de datos para obtener metadatos y a continuación se divide el conjunto de datos a exportar. Cada uno de estos subconjuntos pasará por un trabajo *MapReduce* de solo etapa *Map* para ser copiados en la base de datos externa destino (Figura 62).

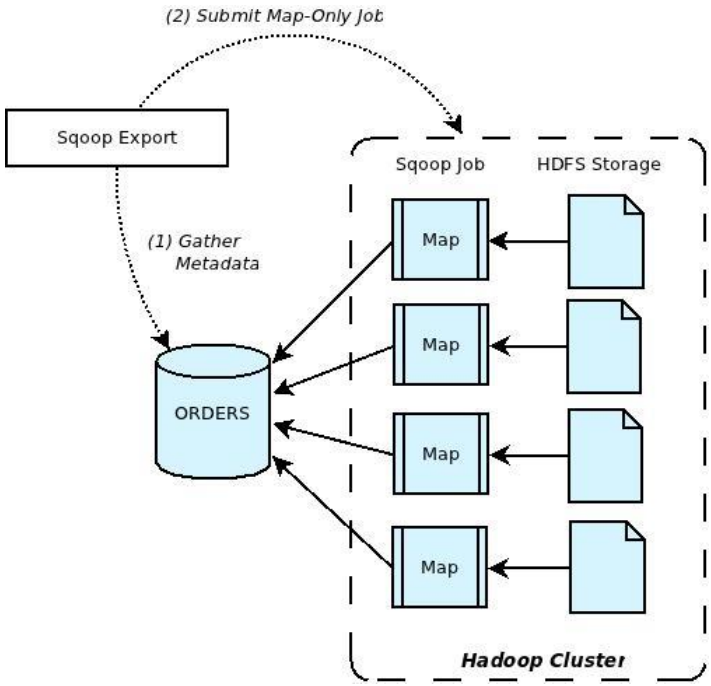


Figura 62. Proceso de exportación en *Sqoop*. Fuente: [176]

A mayores, podemos encontrar otra gran cantidad de herramientas, relacionadas con la gestión de bases de datos, tanto externas como de los almacenamientos que nos podemos encontrar en *Hadoop*, que en principio no parecen necesarias para este trabajo, o por lo menos existen otras alternativas mejores para realizar la misma función que utilizar *Sqoop*.

2.2.3.2.8. *Apache Oozie*

Apache Oozie [177], [178] consiste en una aplicación *web Java* usada para planificar trabajos en *Hadoop*. Para ello, combina múltiples trabajos secuenciales o en paralelo en una única unidad de trabajo. A los correspondientes trabajos a realizar les denomina acciones, soportando acciones *MapReduce, Pig*, *Hive*, *Sqoop*, *Spark*, programas *Java* y *script* de *Shell* entre los más importantes, tanto de forma individual como combinando varios de ellos secuencialmente o en paralelo.

¿Cómo funciona *Oozie*?

En primer lugar, se distinguen dos tipos de trabajos [178]–[180]:

- ***Workflows*** o flujos de trabajo. Estos consisten en grafos acíclicos directos (*DAG*) en los que se especifican la secuencia de acciones a ejecutar. Por ejemplo, en la Figura 63 se muestra un *DAG* simple de una única etapa denominada *"map-reduce wordcount"*. El resto de elementos del diagrama son puntos de control como inicio, finalización y error en el flujo.

- ***Coordinators*** o coordinadores. Permiten definir y controlar calendarios de ejecución de los flujos de trabajo configurados, indicando inicio de ejecución de la tarea, tiempos de repetición de las tareas, número de repeticiones, finalización de repetición de tareas, etc.

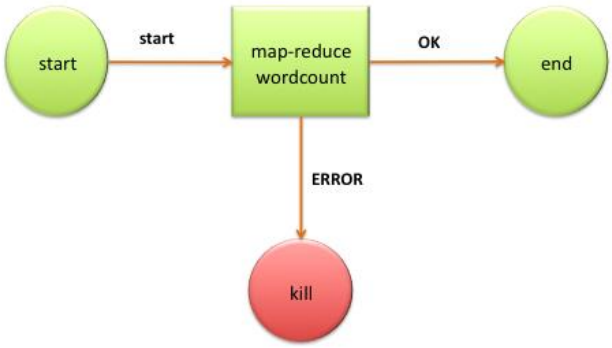


Figura 63. Ejemplo de diagrama DAG de *workflow* en *Oozie* I. Fuente: [180]

En cualquiera de los casos, los nodos de control definen la cronología del trabajo, estableciendo las reglas de inicio y finalización del *workflow*, es decir, *Oozie* se encarga de controlar el camino de ejecución de los trabajos (bifurcaciones, uniones, decisiones), lanzando las acciones que lo componen pero es *Hadoop/MapReduce* quien las ejecuta. De esta forma, *Oozie* no tiene que preocuparse de proporcionar todas las funcionalidades de computación distribuida que proporciona *Hadoop/MapReduce*.

Tanto *workflows* como *coordinators* se especifican mediante un lenguaje propio *hPDL*, bastante parecido a *XML*.

En la Figura 64 puede verse un diagrama DAG de *workflow* de *Oozie* más complejo, en el que se pueden apreciar la ejecución secuencial de acciones de distinto tipo, también hay una ejecución en paralelo o nodos de decisión sobre si ejecutar o no acciones.

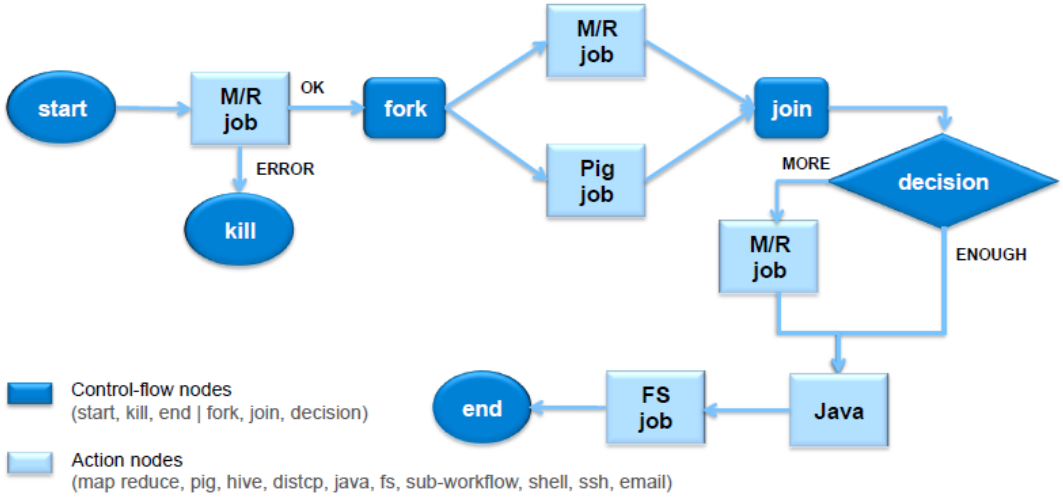


Figura 64. Ejemplo de diagrama DAG de *workflow* en *Oozie* II. Fuente: [181]

2.2.3.2.9. *Apache Ambari* y *Hue*

Apache Ambari [182], [183] es un proyecto cuyo objetivo es hacer más simple la gestión de *Hadoop* desarrollando *software* para el aprovisionamiento, gestión y seguimiento de los *clusters Apache Hadoop*. Para ello, proporciona una interfaz de usuario tipo *web*, que hace uso de las *APIs REST* que se ofrecen para acceder a las distintas tecnologías integradas en *Hadoop*.

¿Qué permite hacer *Ambari*?

Provisión del *cluster Hadoop*. A través de la interfaz *web* de *Ambari* se pueden instalar nuevos servicios[6] en los nodos que componen el *cluster* así como configurar cada uno de estos servicios.

Gestionar el *cluster Hadoop*. *Ambari* proporciona un sistema de gestión que permite iniciar, detener y reconfigurar los servicios *Hadoop* a lo largo del *cluster*.

Monitorizar el *cluster Hadoop*. La parte más interesante de *Ambari* es el panel de mandos (*Dashboard*) ya presentado en la Figura 33. A través de él se puede analizar el estado del *cluster Hadoop* pudiendo configurar qué métricas queremos que se muestren en la página principal. Para ello, presenta un sistema de colección de métricas, denominado *Ambari Metrics System* con el que recoge datos de los distintos servicios y nodos, y con ellos genera gráficos para que el usuario pueda consultar el estado del *cluster*. También proporciona un sistema de alertas, para advertir al usuario de situaciones que requieren de su atención.

A mayores de la parte de monitorización también ofrece interfaces para utilizar los servicios de *Hadoop* de forma alternativa a realizarlo por línea de comandos. A este conjunto de interfaces se les denomina *Ambari User Views* [184]. En la Tabla 10 se detallan las vistas de usuarios que se ofrece *Ambari*.

[6] *Apache Ambari* entiende como servicio *Hadoop* cada una de las tecnologías integradas en la infraestructura *Hadoop*.

User View	Descripción
Tez	La vista de *Tez* (Figura 65) permite visualizar los distintos trabajos lanzados (incluso finalizados) sobre el motor de ejecución *Tez*, ver su progreso, detenerlos, consultar el gráfico *DAG* generado con el programa de usuario, etc.
Hive	La vista *Hive* (Figura 66) ofrece un editor donde escribir y ejecutar consultas *HiveQL* sobre los datos del *cluster*. También permite acceder al historial de consultas lanzadas y la opción de lanzar las consultas sobre *Tez* así como una vista de los diagramas *DAG* generados en *Tez* para ejecutar la consulta.
Pig	La vista de *Pig* (Figura 67) consiste en un editor con el que escribir y lanzar consultas. Permite guardar *scripts*, cargar *UDFs* y ofrece una serie de plantillas con sentencias básicas.
HDFS	La vista de *HDFS* (Figura 68) permite navegar por los ficheros guardados en este sistema de ficheros, editarlos, cambiar permisos, crear ficheros/directorio, subir ficheros desde la máquina que lanza el navegador *web,* etc., todo en función de los permisos que el usuario registrado en la interfaz tenga.
Ficheros Locales	Similar a la vista de *HDFS* pero sobre el sistema de ficheros del SO donde se instaló *Hadoop*, por ejemplo, la máquina virtual de la *sandbox* de *Hortonworks*.

Tabla 10. Vistas de usuario de *Ambari*

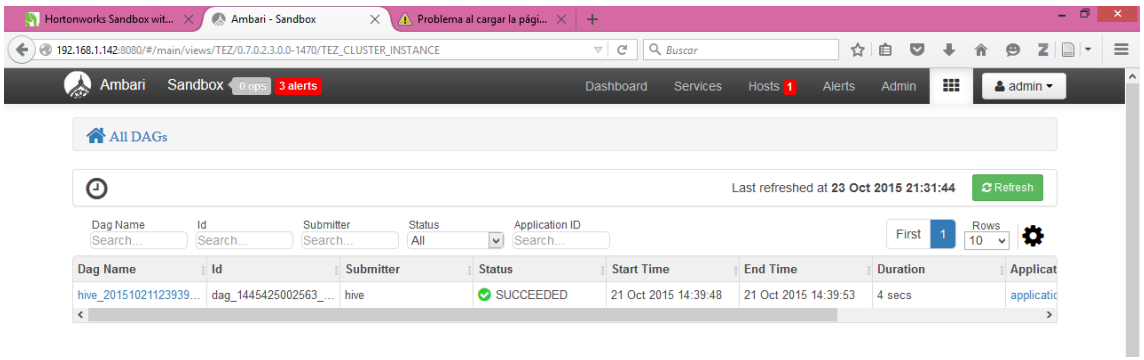


Figura 65. *Apache Ambari* – Vista de Usuario: *Tez*

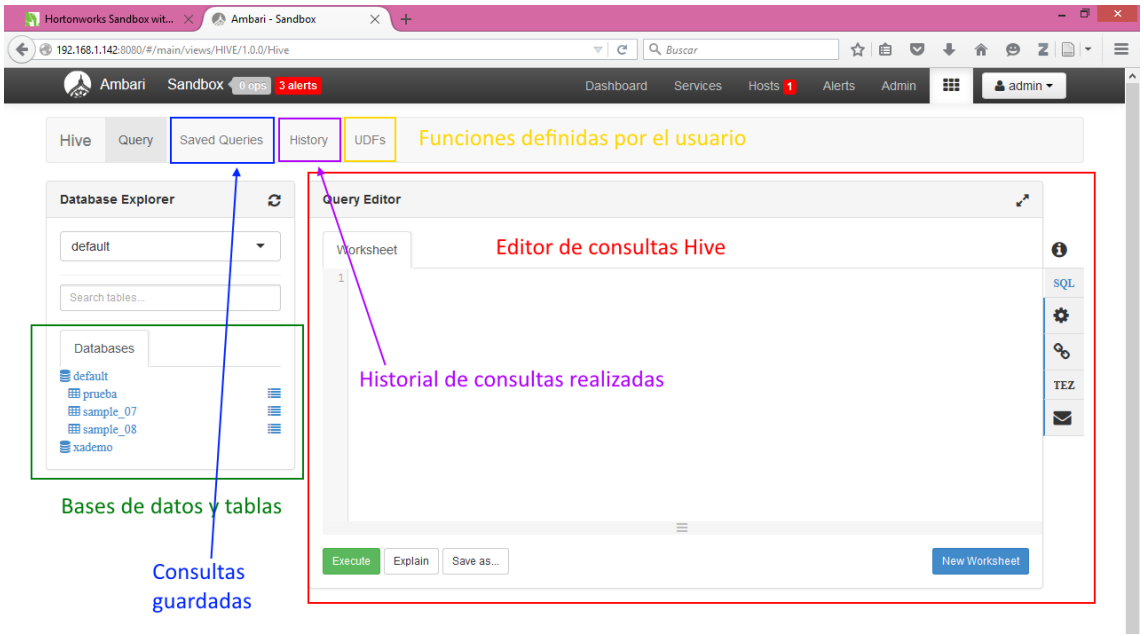


Figura 66. *Apache Ambari* – Vista de Usuario: *Hive*

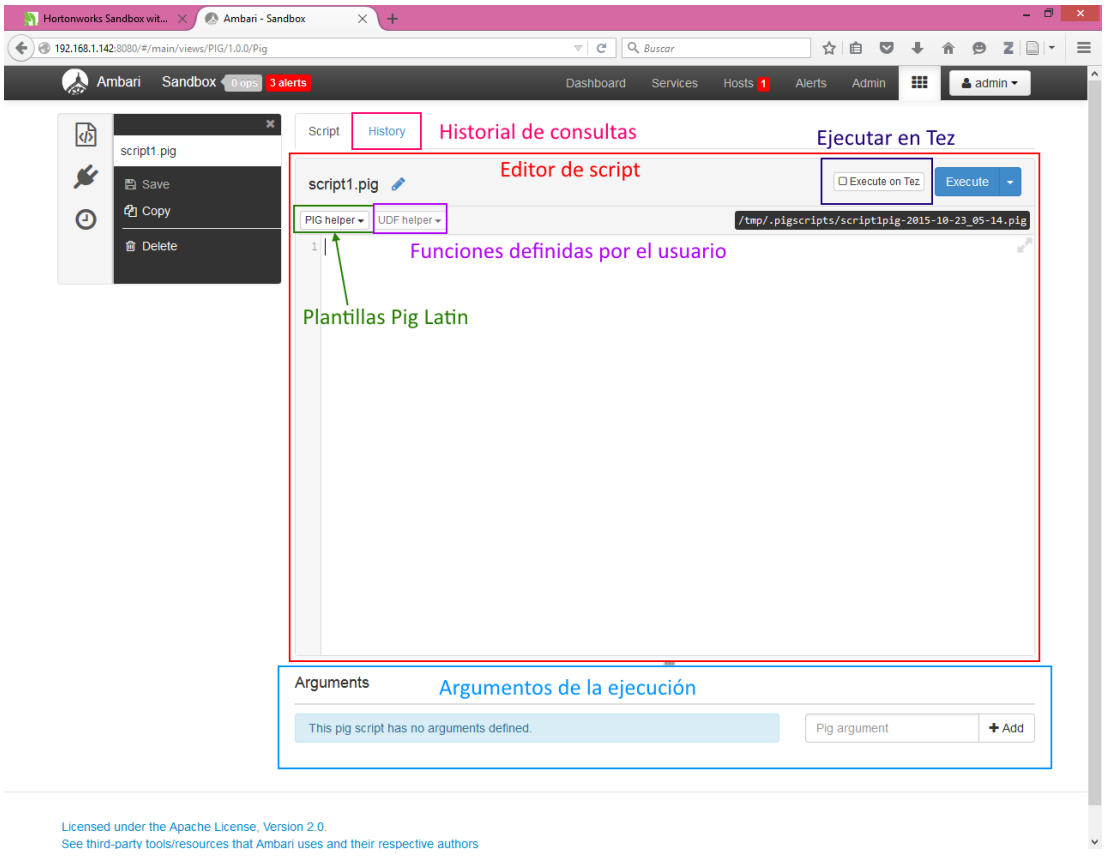


Figura 67. *Apache Ambari* – Vista de Usuario: *Pig*

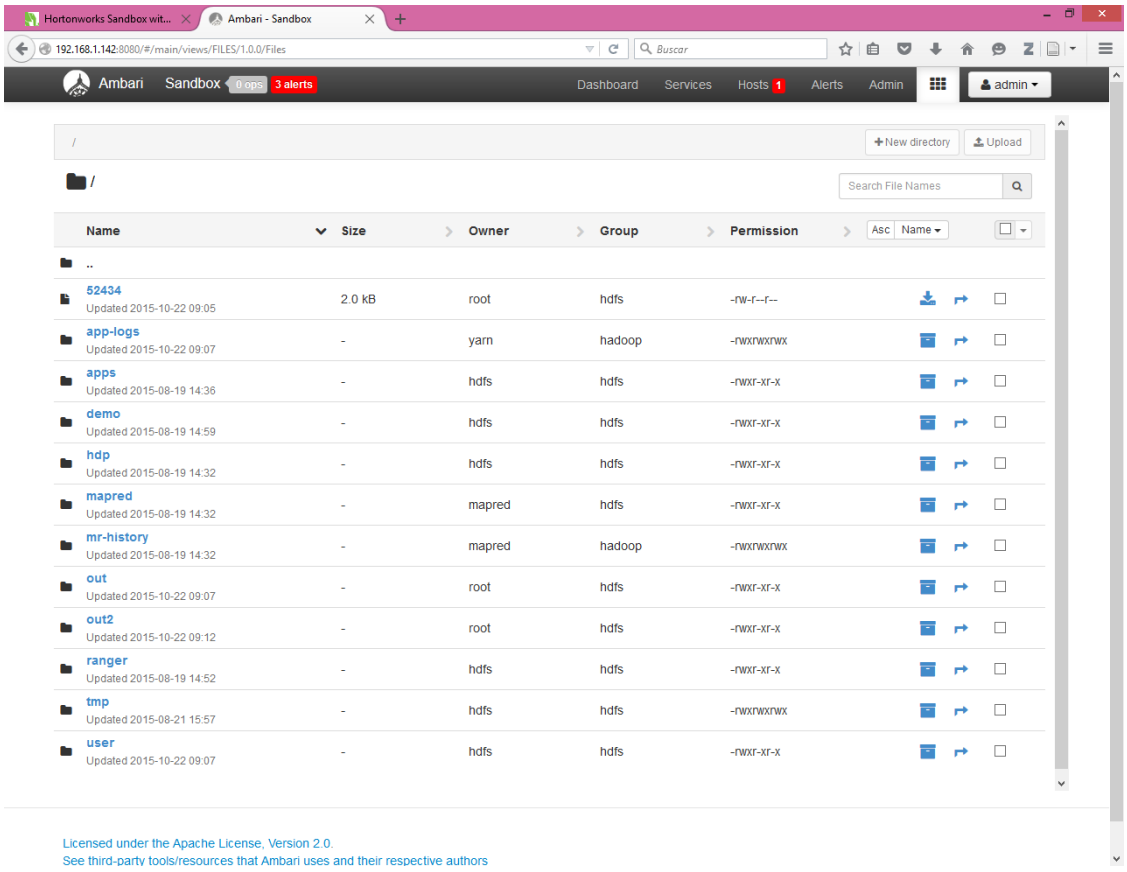


Figura 68. *Apache Ambari* – Vista de Usuario: *HDFS*

Aprovechamos este apartado para hablar de ***Hue*** [185]. Este es una interfaz *web* de usuario que permite utilizar los servicios *Hadoop* de forma alternativa a la línea de comandos. Como *Ambari*, presenta vistas con editores *Hive*, *Pig* y acceso al sistema de ficheros *HDFS*, todas ellas muy similares a las ya mostradas de *Ambari*. La principal diferencia está en el editor *Oozie*. *Hue* presenta un editor de *workflows* y *coordinators* de *Oozie*, lo cual es una enorme ventaja. Utilizar *Hive*, *Pig* o navegar por *HDFS* puede ser igual de fácil mediante la herramienta de línea de comandos o por la interfaz de *Ambari*/*Hue*. No ocurre lo mismo con *Oozie*. Para definir *workflows* y *coordinators* se deben generar ficheros en formato *hPDL* con una estructura y anidaciones muy concretas, con necesidad de definir cada uno de los puntos de los diagramas *DAG* característicos. Esto puede ser muy tedioso y complicado de elaborar manualmente. El editor *Oozie* de *Hue* proporciona una forma muchísimo más sencilla de realizar esta configuración.

Los *workflows* se definen mediante un editor con un aspecto bastante gráfico ya que en él se van seleccionando el tipo de acciones que queremos realizar y vamos especificando su ejecución secuencial o en paralelo arrastrando el tipo de acción hasta el punto de ejecución (Figura 69 y Figura 70), para después configurar el funcionamiento concreto de esa acción rellenando un formulario con los datos necesarios para ejecutar ese tipo de acción (Figura 71). La presentación difiere si utilizamos *Hue* 2.6 o 3.8 pero la esencia es la misma.

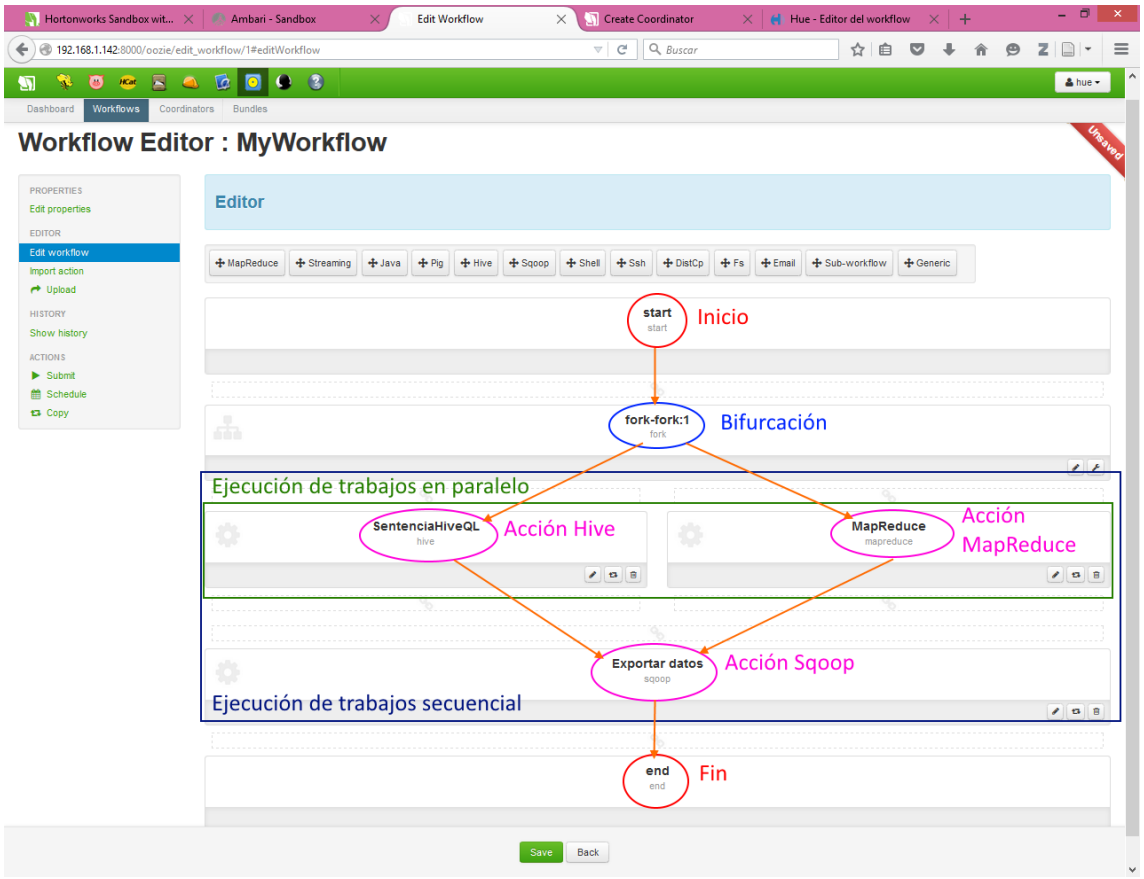


Figura 69. Definición de *workflows Oozie* en *Hue* 2.6

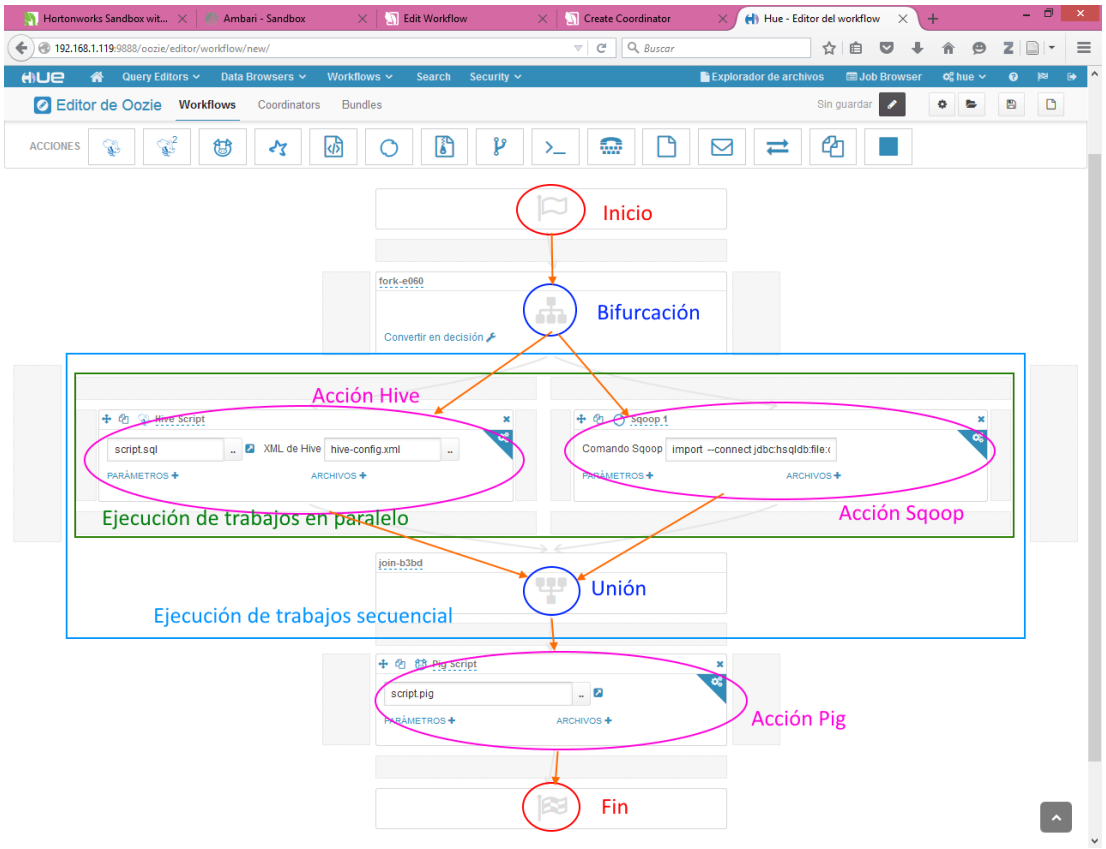


Figura 70. Definición de *workflows Oozie* en *Hue* 3.8

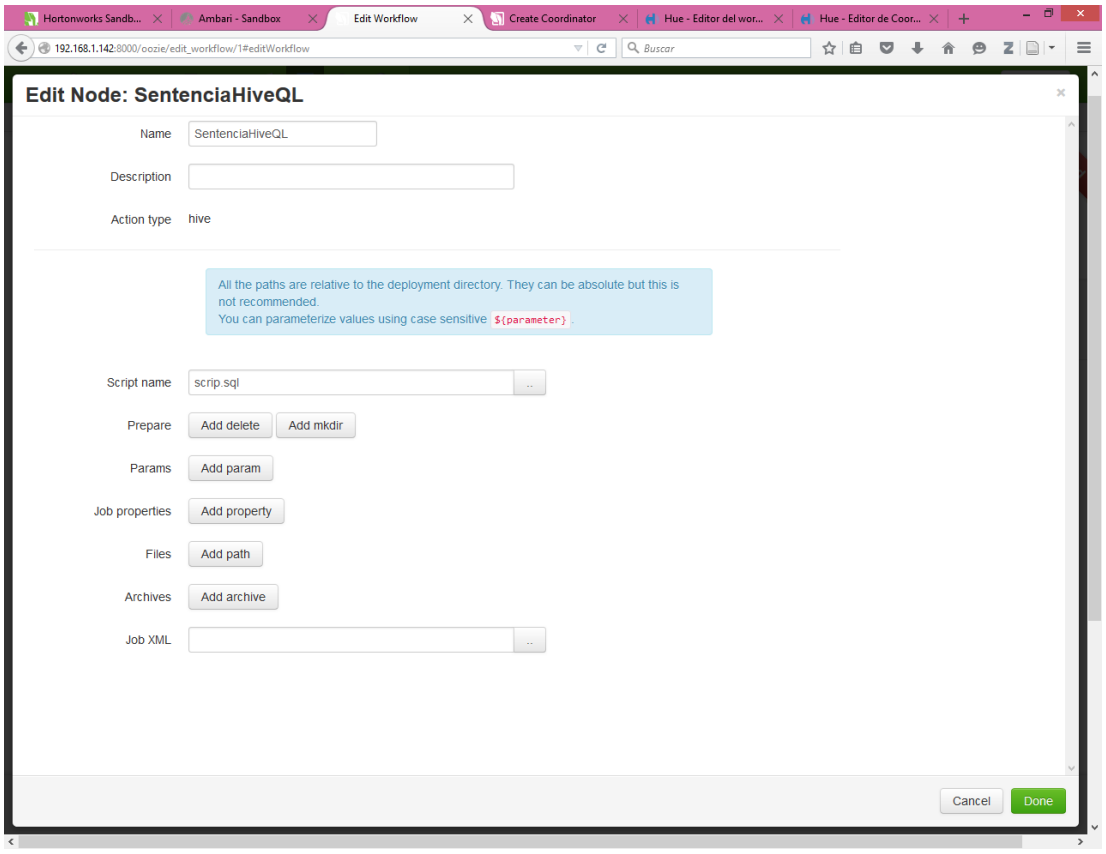


Figura 71. Formulario definición acción *Hive* en *Oozie*

Los *coordinators* se definen por medio de otro formulario en el que se especifican las características temporales de planificación de ejecución del *workflow* seleccionado (Figura 72 y Figura 73). El formulario a rellenar es ligeramente distinto en *Hue* 2.6 y 3.8.

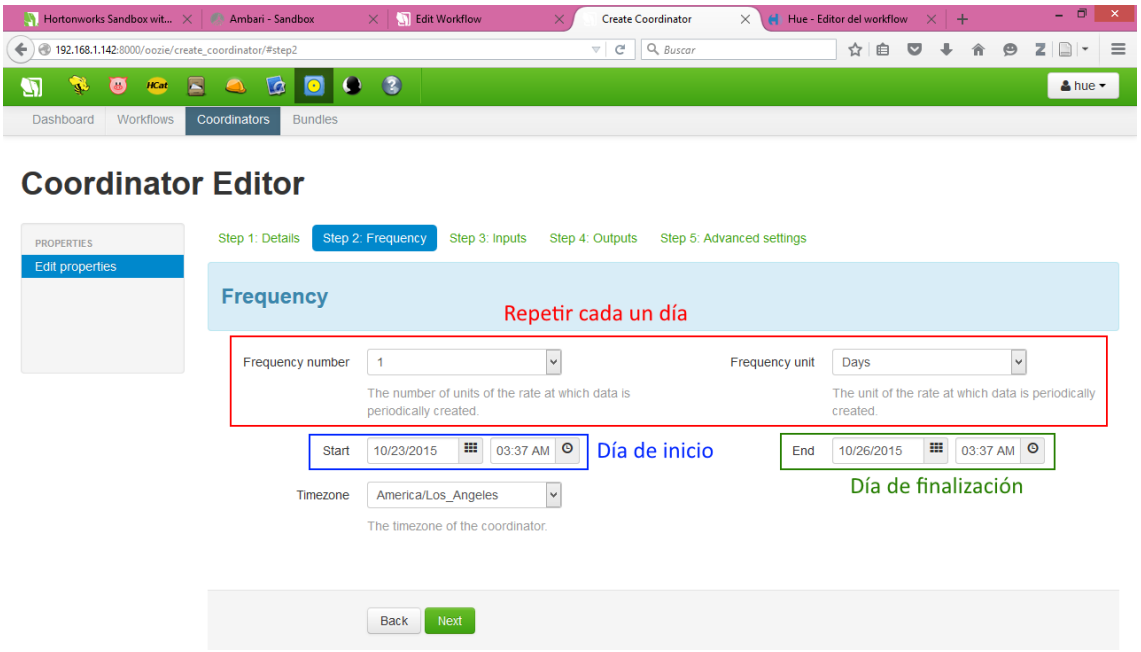


Figura 72. Definición de *coordinator Oozie* en *Hue* 2.6

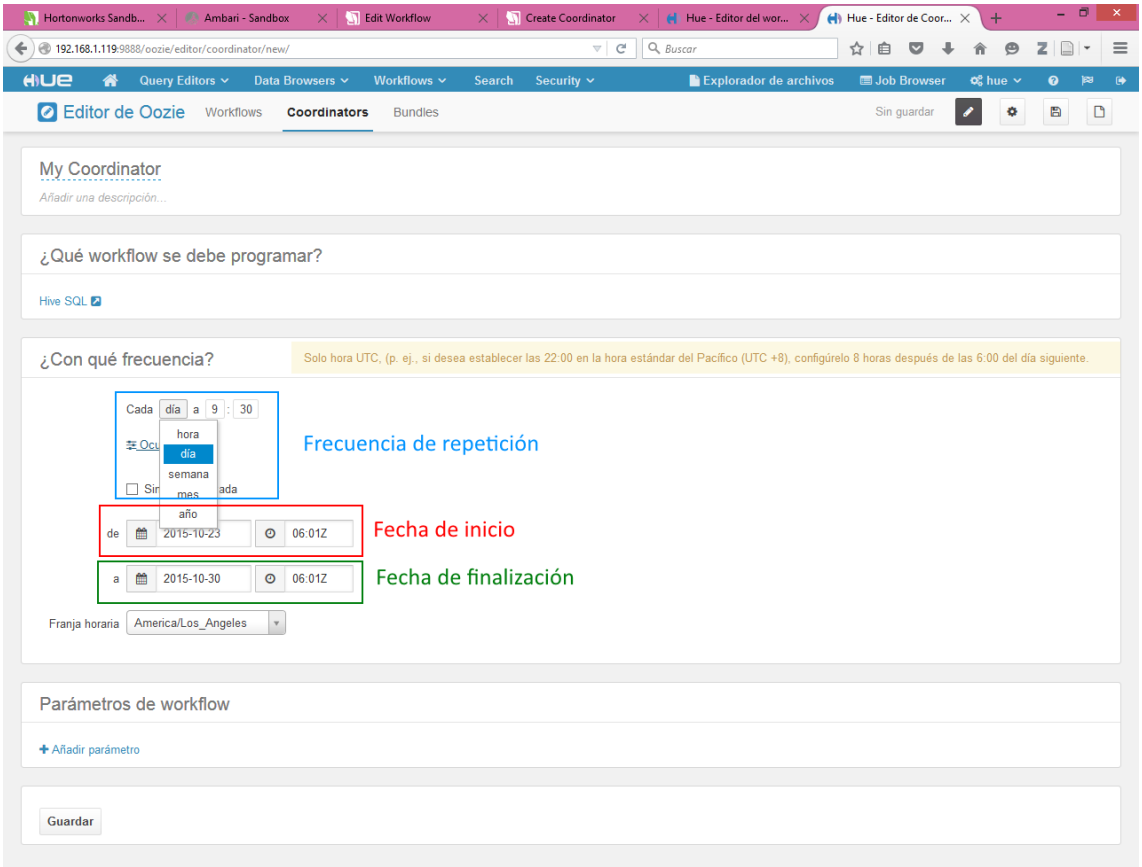


Figura 73. Definición de *coordinator Oozie* en *Hue* 3.8

En cualquiera de los dos casos, *Hue* nos proporciona el fichero *workflow* o *coordinator* que deberíamos haber generado manualmente para producir el mismo efecto, de tal forma que lo podemos llevar a cualquier otra infraestructura *Hadoop* o que soporte el uso de *Oozie* y ejecutarlo mediante línea de comandos.

A parte de editores para la definición de trabajos *Oozie*, *Hue* también permite monitorizar su estado, como por ejemplo se puede ver en las Figura 74 y Figura 75.

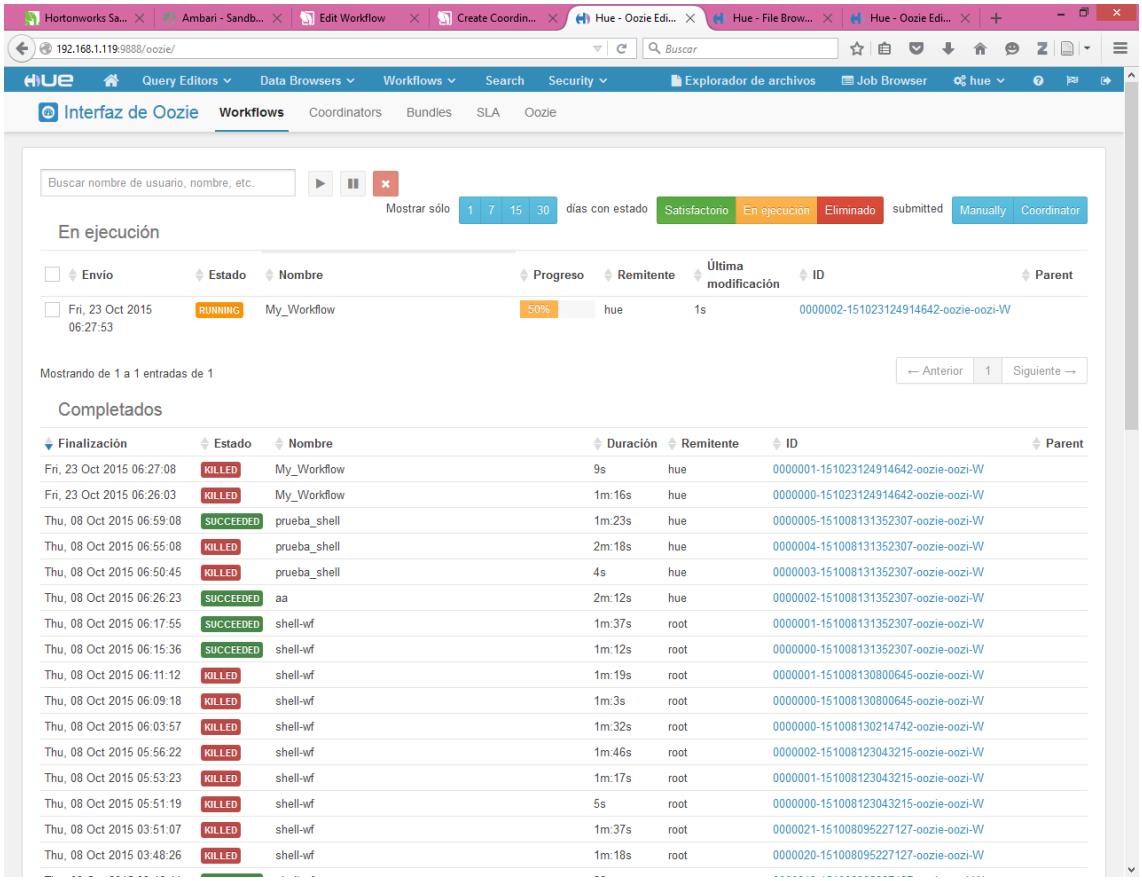


Figura 74. Panel de visualización de estado de los trabajos *Oozie* lanzados

Por lo demás, recordar la comparativa entre Hue 2.6 y 3.8 en la Sección "Distribuciones de Hadoop: Hortonworks vs Cloudera". La versión 3.8 es más completa ya que proporciona editores para *Spark*, *Sqoop*, mayor número de acciones configurables por interfaz en *Oozie*, etc. Como se pueden tener varias versiones de *Hue* instaladas simultáneamente, no hay más que escoger en cada momento aquella que nos pueda ser de mayor utilidad o que nos permita realizar algo de forma más sencilla. El resto de interfaces no explicadas no son de mayor utilidad en el desarrollo de este trabajo.

2.2.3.2.10. *Apache ZooKeeper*

Apache ZooKeeper [186], [187] consiste en un sistema de alta disponibilidad para coordinar procesos distribuidos y cuya principal responsabilidad es la sincronización de los distintos nodos del *cluster*. Las aplicaciones distribuidas usan *ZooKeeper* para almacenar y mediar cambios importantes en la información de configuración. Proporciona una interfaz y servicios muy simples. Beneficios:

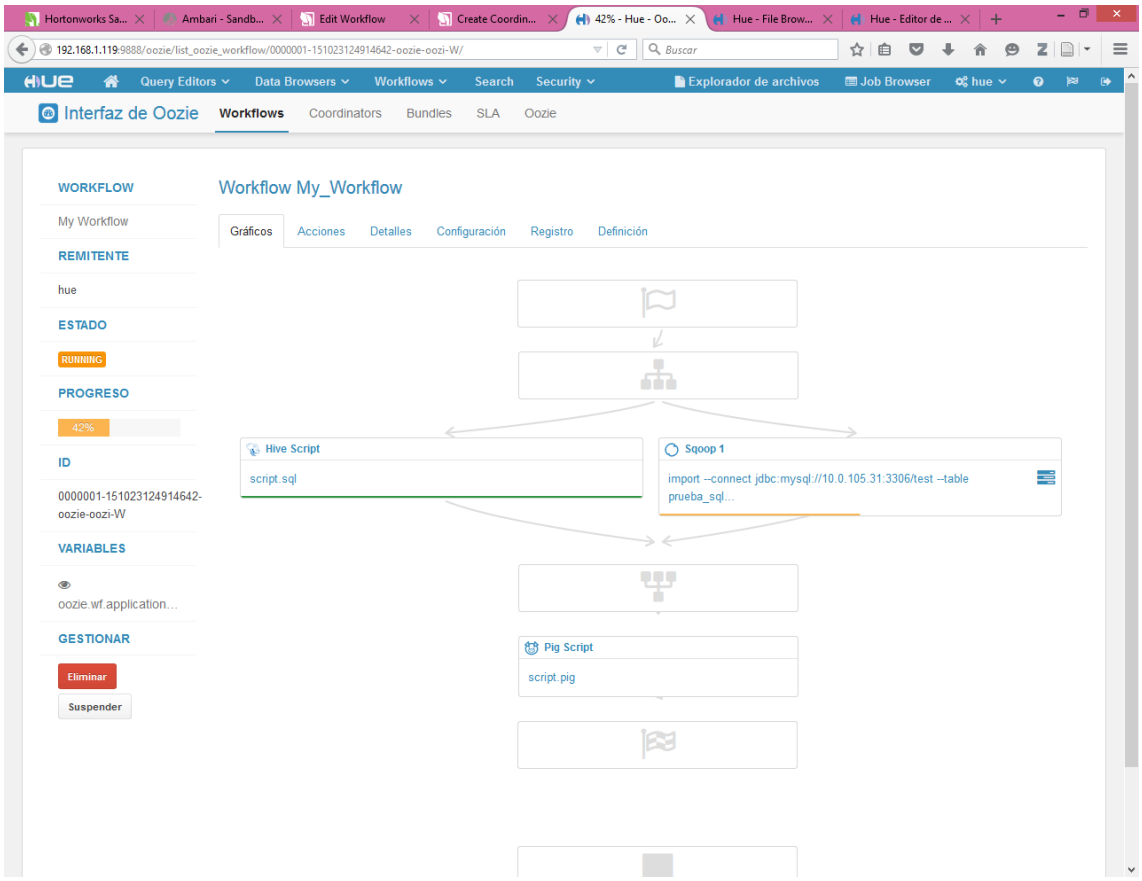


Figura 75. Detalle del estado de un *workflow Oozie*

- Rapidez. Es especialmente rápido con cargas en las que es más común leer que escribir datos. El ratio lectura/escritura ideal es 10:1.
- Fiabilidad. *ZooKeeper* se replica a través de varios nodos y los distintos servidores se conocen entre sí. No hay ningún punto crítico de fallo, porque mientras haya un número mínimo de servidores disponible, el servicio *ZooKeeper* también lo estará.
- Simplicidad. Mantiene un espacio de nombres jerárquico estándar, similar a una estructura directorio – ficheros.

2.2.3.2.11. *Visualización de resultados*

La etapa de visualización de resultados se encargará de extraer los datos generados en *Hadoop* y generar con ellos gráficos que sean de interés para el usuario. *Hadoop* no proporciona herramientas integradas que implementen estas funcionalidades. Parece que *Apache Solr o Apache Zeppelin* realizan algo parecido, pero no con el suficiente detalle de análisis que necesitamos, por ello vamos a optar por usar la combinación *ElasticSearch* y *Kibana*. Entre otras muchas alternativas hemos escogido esta por ser ambas herramientas gratuitas y de entre las más populares. Además, *Kibana* está orientado al análisis de registros temporales, una característica que cumplirán los datos resultado de nuestro análisis, para poder filtrar los resultados que se muestran en pantalla según cuando se produjeron. La estrategia va a consistir en extraer los datos de *Hadoop* y llevarlos, como paso intermedio, a *ElasticSearch*. *Kibana* nos permitirá generar multitud de gráficos distintos con esos datos.

2.2.3.2.11.1. *ElasticSearch*

ElasticSearch [188], [189] es una base de datos *NoSQL* orientada a documentos en formato *JSON* y basada en *Apache Lucene*. Es una herramienta ampliamente utilizada en motores de búsquedas de texto en documentos de datos, proporcionando funcionalidades con muy baja latencia, dado que los datos están indexados. *ElasticSearch* permite configurar un *cluster* con distintos nodos a través del cual se distribuirán los datos, para después realizar búsquedas sobre ellos. En principio, no nos centraremos mucho en este aspecto, porque no pretendemos montar un *cluster ElasticSearch*, ya que en tan solo un nodo almacenaremos todos los resultados obtenidos mediante *Hadoop*. En todo *cluster* debe haber un nodo de datos y un nodo maestro, que en nuestro caso será el mismo. El nodo de datos se encargará de almacenar los datos y ejecutar las consultas. El nodo maestro es el que se encarga de dirigir el *cluster*, ordenando la ejecución de consultas, recuperando índices corruptos, etc.

Los datos se introducen en *ElasticSearch* en índices y las búsquedas se restringen a un único índice, es decir, no se pueden realizar búsquedas en las que queramos que los resultados provengan de dos índices distintos. Esto da muchas veces lugar a redundancia de datos ya que cada índice se genera exclusivamente a partir de un documento. Para evitarlo, se pueden definir alias, bajo los cuales se agrupan varios índices, de tal forma que realizando una búsqueda sobre un alias, se pueden realizar búsquedas sobre varios índices a la vez.

El almacenamiento de los datos en los nodos es gestionado directamente por el *cluster*, por medio de dos parámetros configurables por el usuario. El primero se denomina *Shard* y hace referencia al número de partes en que se dividirá un conjunto de datos para repartirlo por los nodos de datos del *cluster*. El segundo parámetro se denomina *Factor de Replicación*, con el que se especifica cuantas réplicas en otros nodos se hará de cada uno de los *shard's,* para no perder datos en caso de fallo en alguno de los nodos de datos. Por ejemplo, en la Figura 76 se muestra el esquema de un *cluster ElasticSearch* con tres nodos de datos a través de los cuales se almacena un índice, el cual se ha dividido en 3 *shards*. Cada uno de estos *shards* se ha distribuido por los tres nodos, y, a mayores, por cada *shard* principal se ha generado una réplica. Las réplicas de estos *shards* también se han distribuido, pero por nodos distintos al nodo con el *shard* principal.

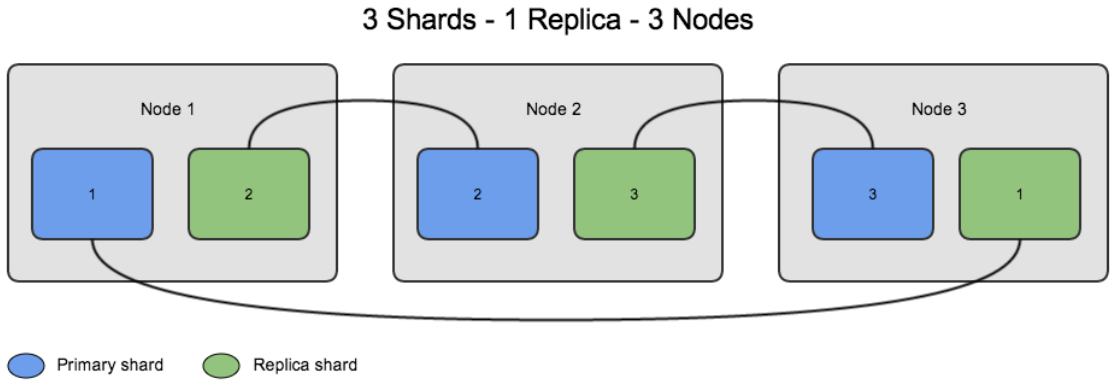


Figura 76. Almacenamiento de datos en *shards* y réplicas en *ElasticSearch*. Fuente: [190]

ElasticSearch se gestiona mediante una *API REST*, así que todas las peticiones se realizarán por medio de métodos *HTTP*. Estas peticiones tendrán por objetivo añadir índices, añadir datos a un índice, borrar un índice (nunca elementos individuales de un índice), consultar datos de un índice, etc. Además, los índices almacenados se podrán consultar por medio de un navegador *web* accediendo al servidor *ElasticSearch* y navegando por la jerarquía de índices. A mayo-

res, *ElasticSearch* tiene la posibilidad de añadir *plugins* que proporcionan una interfaz gráfica *web* con la que realizar las peticiones sobre *ElasticSearch*, navegar sobre los índices añadidos o consultar cada una de las entradas de los índices, por ejemplo, usaremos *Plugin Head*, una interfaz muy simple que nos permite consultar el estado del *cluster*, los índices creados, los datos insertados en cada índice o realizar peticiones. Por ejemplo, en la Figura 77 podemos ver la página principal de este *plugin*. En ella aparece una entrada por cada nodo que compone el *cluster ElasticSearch*. En la primera entrada, cada columna representa un índice cargado en *ElasticSearch*. En cada uno de los cruces nodo – índice se indica que *shard* de ese índice se almacenan en ese nodo.

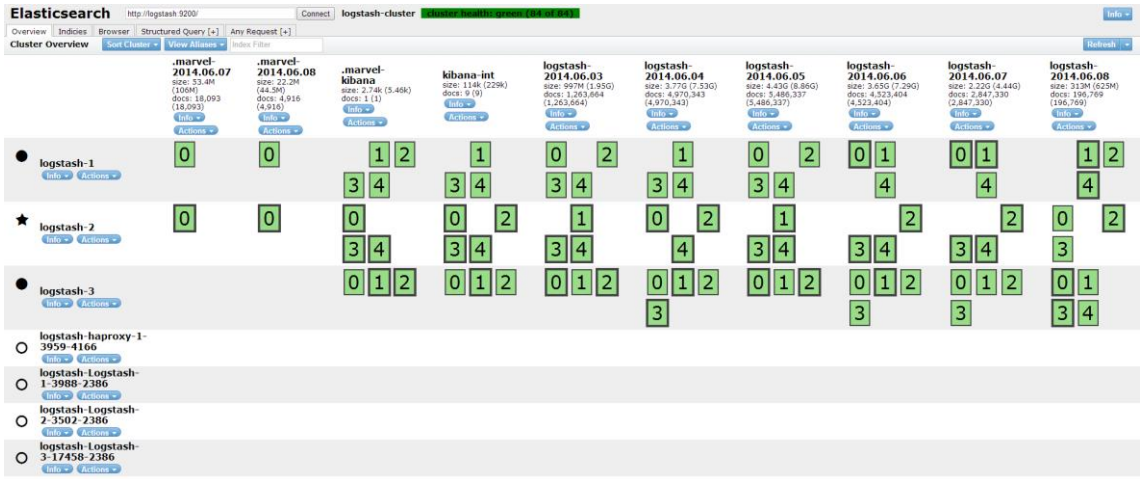


Figura 77. Ejemplo de página principal de *Plugin Head*. Fuente: [191]

En nuestro caso, solo necesitaremos añadir información a *ElasticSearch* ya que toda la parte de búsquedas la realizará directamente *Kibana* tras una simple configuración. Pero la adicción de información a *ElasticSearch* tampoco la vamos a llevar a cabo de forma manual por medio de la *API REST* o la interfaz *Plugin Head*. Existen una serie de conectores que permiten cargar datos en *ElasticSearch* desde otras herramientas. Por ejemplo, en nuestro caso usaremos un conector *Hadoop* que permite integrar *Hive* y *ElasticSearch* [192] (Figura 78) para introducir datos en los índices de *ElasticSearch* mediante sentencias *Hive* y cualquier editor de consultas, como los que nos proporcionan *Apache Ambari* o *Hue*, lo cual, entre otros, nos evita tener que generar documentos *JSON* con los datos correspondientes. La única operación que deberemos realizar directamente sobre *ElasticSearch* es la creación del índice.

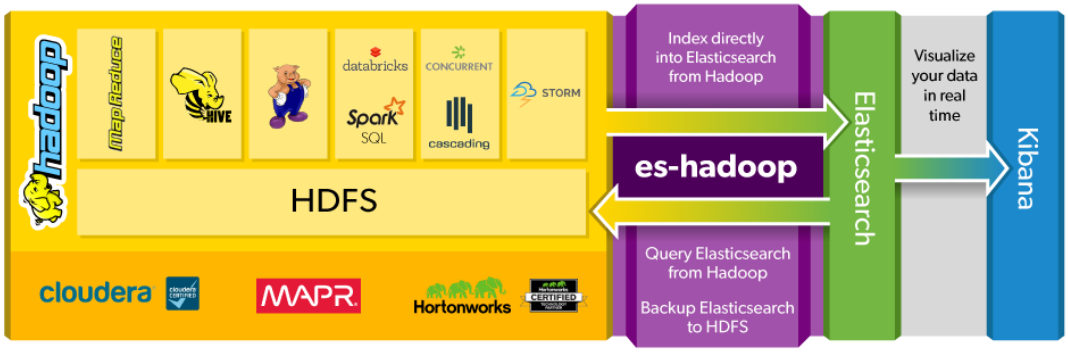


Figura 78. Integrar *Hadoop* y *ElasticSearch*. Fuente: [192]

2.2.3.2.11.2. *Kibana*

Kibana [193] es una herramienta de análisis de datos exclusivamente integrada con *ElasticSearch*, es decir, solo permite analizar datos almacenados en *ElasticSearch,* centrada en analizar ficheros de registros temporales.

Kibana se puede usar para buscar, ver e interactuar con los datos almacenados en los índices *ElasticSearch*, de tal forma que se puede realizar fácilmente análisis avanzados de datos y visualizar estos datos en gran variedad de gráficos, tablas y mapas, todo ello por medio de una interfaz *web*, como la mostrada en la Figura 79.

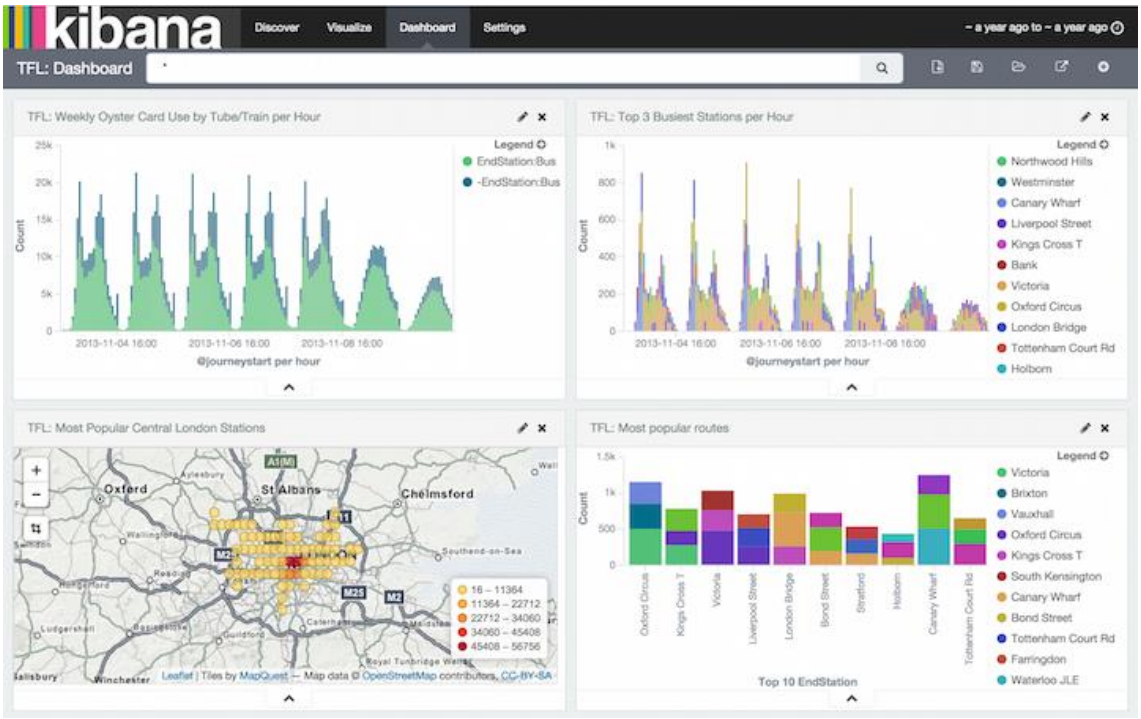


Figura 79. Ejemplo de uso de *Kibana*. Fuente: [194]

2.3. Conclusiones

Una vez finalizado todo el estudio planteado a lo largo de este capítulo debemos responder a la pregunta: ¿y por qué *BigData* para realizar algo que ya se hace mediante otras técnicas, en concreto la Minería de Datos?. La respuesta se puede dar por dos caminos.

En primer lugar, si se revisan detalladamente los estudios analizados en el Apartado 2.1.2, así como en otros tanto no incluidos en esta revisión, los trabajos realizados han considerado conjuntos de estudiantes pequeños en tamaño (con cerca de 1000 estudiantes en el caso más numeroso y entre 10 y 20 cursos como máximo). Incluso se han encontrado estudios que en el propio artículo incluían los datos utilizados, no llegando a más de 50 registros. Es decir, se han probado algoritmos con cantidades de datos muy pequeñas. En un caso real, tendríamos cantidades enormes de datos. En nuestro caso, la Universidad de Valladolid tiene cerca de 25000 estudiantes y otros miles de profesores, lo que dará lugar a cientos de miles de datos a analizar. Por ejemplo, solo el rastreo de la actividad en un curso de la ETSIT ha dado lugar a cerca de 36000 registros de actividad, a los que habría que sumar toda esa información administrati-

va a mayores. Esto implica, que en la realidad tendremos que hacer frente a cantidades enormes de datos, podría considerarse normal alcanzar hasta el millón de registros. La Minería de Datos tiene mecanismos y algoritmos probados, pero no se sabe hasta qué punto el tratar con cantidades enormes de datos puede ser exitoso. Por ello, podría ser conveniente utilizar una infraestructura *BigData* para garantizar la obtención de un resultado, independientemente del tiempo necesario para ello.

En segundo lugar, aunque la Minería de Datos fuera capaz de resultar exitosa en cualquier análisis, si necesitamos hacer frente a un trabajo de Minería de Datos de enormes dimensiones, necesitaremos tras ello una infraestructura *hardware* potente, lo cual se traduce en un coste considerable. Una infraestructura *BigData* será mucho más económica, al no requerir una infraestructura *hardware* demasiado estricta, por lo general los únicos requerimientos son la memoria RAM y el almacenamiento. El almacenamiento en disco es hoy, relativamente, barato. La memoria RAM es más cara, pero tampoco se requieren grandes cantidades, por ejemplo, en nuestro caso implementaremos una infraestructura *BigData* compuesta por una máquina virtual por medio de un simple ordenador con 32 GB de RAM.

Capítulo 3

Análisis del problema

En este capítulo se analizará cuál es el problema que se pretende resolver en la realización de este trabajo, en esencia, resumir brevemente la información del Capítulo 2 para centrar los objetivos. En capítulos anteriores se indicó que este trabajo pretendía aplicar conceptos de *BigData* para analizar la actividad que alumnos y profesores desencadenan sobre la plataforma educativa *Moodle*. Pero, ¿qué se pretende realizar en concreto? *Moodle* ya proporciona una funcionalidad de cálculo de estadísticas, por lo que no tendría sentido dedicar *BigData* solo al cálculo de estadísticas. Así, se necesita algún objetivo más.

En el apartado 2.1.2 se ha presentado un extenso análisis de otros estudios con objetivos similares, utilizando técnicas de *Data Mining*. Al finalizar el capítulo anterior ya se comentó el dudoso éxito de la Minería de Datos cuando tiene que enfrentarse a grandes cantidades de datos, como las que pretendemos hacer frente en este trabajo. Por ello, parece interesante aplicar técnicas de *BigData* para conseguir objetivos similares.

A la vista de todos los ejemplos enumerados en el apartado 2.1.2, se ha comprobado cuáles son los algoritmos más utilizados:

- Reglas de asociación: *Apriori*.
- Clasificación: Árbol de decisión C4.5.
- *Clustering: K – Means*.

Pero a la hora de tratar de replicar algunos de estos estudios en *BigData* deberemos tener en cuenta las posibilidades que tenemos. Nuestra infraestructura *BigData* ofrecerá dos librerías de algoritmos de *Machine Learning*: *Apache Mahout* y *MLlib* de *Apache Spark*. Discutiremos qué algoritmos necesitamos implementar y qué alternativas a los aquí citados tenemos.

También hemos podido comprobar cuáles son los algoritmos que se suelen emplear según la finalidad concreta:

- Las reglas de asociación se suelen utilizar para implementar sistemas de recomendación.
- Los sistemas de predicción se implementan con algoritmos de clasificación, principalmente árboles de decisión, pero nunca con algoritmos de *clustering*.

Una parte importante es el pre – procesado previo de los datos antes de ser analizados, ya que nos permitirá mejorar los resultados. En este sentido:

- Normalmente los valores numéricos se suelen discretizar. Esta es una etapa con muchos problemas de interpretación ya los datos extremos de cada categoría de discretización son más distintos que los datos que están cerca del umbral que separa dos categorías. Por ejemplo, si dividimos las notas en cuatro grupos (*FAIL* (nota < 5), *PASS* (5 < nota < 7), *GOOD* (7 < nota < 9) y *EXCELLENT* (nota > 9)), un alumno con nota 6.9 caerá en la misma categoría que un alumno con nota 5.1, aun siendo la nota muy distinta. El problema crece si lo comparamos con el alumno que tiene una nota 7.1, ya que las notas apenas se diferencian en el rango continuo, pero en el discreto quedan separadas.

- Dependiendo del resultado del estudio, se pueden eliminar los *outliers* de los datos, o mantenerlos para centrarse en su análisis.

- En ocasiones también puede ser interesante eliminar del análisis aquellos atributos de los datos que menos relevancia tienen.

En base a todo el estudio realizado en este apartado, son varias las ideas que surgen sobre qué cosas interesantes se podrían hacer:

- Análisis estadístico de los datos de los *logs* de *Moodle*:
    - Número de visitas a recursos.
    - Número de visitas a la página principal de curso.
    - Número de sesiones.
    - Acceso desde fuera/dentro de la universidad.
    - Cuándo se accede a *Moodle*.
    - Cuándo acceden los estudiantes a los recursos desde que se publican.
    - Tiempo medio que dedican los estudiantes a una actividad.
    - Relación tiempo/rendimiento al realizar una actividad.
    - Uso de los foros.
    - Fechas de subida de entregas.
- Agrupar estudiantes con características similares de aprendizaje con los que podamos:
    - Diferenciar los alumnos que llevan bien el curso y los que van peor.
    - Diferenciar alumnos que llevan el curso al día con los que aparentemente han abandonado la asignatura.
    - Diferenciar alumnos motivados con la asignatura y los más dejados.
    - Diferenciar alumnos en base a las actividades que han realizado en *Moodle* y la nota final que han conseguido.
    - Diferenciar alumnos en función del tiempo que tardan en realizar actividades y el rendimiento/resultado de estas.
- Implementar mecanismos de predicción:

  - Predecir la nota final que tendrá un estudiante mediante mecanismos de clasificación en base a múltiples métricas, como actividad en *Moodle*, recursos visitados, acceso a los recursos adicionales, visita a los foros.

  - Analizar el número de visitas por día a cualquier punto de la página de curso y predecir si una posible bajada puede ser debida a abandono de estudiantes para identificar qué actividades o en qué punto se encontraba la asignatura para que los estudiantes no se vieran capaces de seguir.

- Detección de anomalías:

  - Detectar alumnos que destacan sobre los demás o aquellos que parecen tener problemas de aprendizaje para darles el apoyo que corresponda en cada caso.

- Generar reglas de asociación con las que podamos:

  - Buscar relaciones (acceso a determinados recursos, realización de ciertas actividades, nota obtenida en actividades o exámenes parciales) entre pasos seguidos por un alumno y resultados finales, como nota.

  - Buscar relaciones entre distintos apartados de las actividades. Por ejemplo, si alguna actividad consiste en resolver preguntas, en principio independientes, buscar relaciones del tipo: si no se contesta adecuadamente esta pregunta, tampoco está otra.

- Sistemas de recomendación:

  - Indicar a un alumno qué otras tareas puede hacer (actividades extras, leer recursos adicionales, leer un mensaje del foro) para mejorar su rendimiento en base a que otros alumnos lo han hecho y les ha servido.

  - Recomendar a un alumno en qué debe matricularse el año siguiente en base a experiencia con otros alumnos en su misma situación en años anteriores (en base a la nota que sacaron cuando sí se matricularon).

- Clasificar los cursos en función de:

  - La actividad que hay en ellos.

  - La relación entre recursos de apuntes/recursos extras.

  - Del porcentaje de alumnos aprobados.

  - El porcentaje de alumnos que parecen haber dejado la asignatura por no acceder al curso.

- Clasificación de las actividades en *Moodle*:

  - En función de cómo han contribuido a la nota final de los estudiantes.

  - En función de sus visitas.

- Diferencias en el uso de *Moodle* por parte de estudiantes que completaron el curso (tanto si aprobaron como si suspendieron pero se presentaron) y aquellos que abandonaron la asignatura.

- Patrones:
    - Sobre secuencias típicas que siguen los alumnos a lo largo de las sesiones.
    - Patrones sobre cuando cada estudiante suele acceder a *Moodle* (durante las horas de clase, al mediodía, por la noche, a cualquier hora). Podría ponerse en el apartado de clasificar los estudiantes en función de su hora de acceso a *Moodle*.

No solamente deberemos tener en cuenta qué nos permite hacer nuestra infraestructura *BigData*, sino también con qué datos contamos para hacer el análisis, ya que puede ocurrir bien no podamos extraer campos necesarios para realizar estos análisis o bien no tengamos datos suficientes. En este segundo caso nos podemos plantear el generar datos ficticios que contengan toda la información que necesitamos para que el algoritmo trabaje, y ya más adelante se podría valorar el sentido de estos resultados. Obviamente, si generamos manualmente datos, los resultados serán, con mucha probabilidad, incoherentes. La coherencia de los resultados debería evaluarse con datos reales. Además, se ha comprobado que los estudios no se centran solo en analizar los registros de actividad de usuarios de *Moodle*, si no que requieren de información adicional, lo que en muchos casos se ha denominado información administrativa o académica. Esto implica que deberemos ampliar nuestro estudio para obtener también estos datos y poder realizar un trabajo más completo.

Con el objetivo de ofrecer un primer acercamiento al uso de *BigData* y así poder encontrar sus puntos fuertes y débiles en la práctica, de entre los estudios analizados en el apartado 2.1.2 nos vamos a centrar en dos:

- *"Mining Moodle to understand Student Behaviour*" de Casey, Gibson y Paris [10].
- *"Data Mining in course Management Systems: Moodle case study an tutorial"* de Romero, Ventura y García [15]

Los requisitos principales de este trabajo serán replicar los estudios aquí citados adaptándolos a nuestras herramientas *BigData*.

El primer estudio nos llevará a buscar influencias entre parejas de parámetros, es decir, calcular la influencia que cierto comportamiento en *Moodle*, por ejemplo, el número de veces que un usuario accede al contenido de *Moodle*, tiene sobre la nota que ha obtenido el alumno.

Siguiendo el segundo estudio, aplicaremos técnicas de *Machine Learning* a través de *BigData* (*Apache Mahout*), en concreto:

- Un mecanismo de *Clustering* para el agrupamiento de usuarios en tres categorías. Preferentemente mediante el algoritmo de *K – Means*.
- Un algoritmo de Reglas de Asociación para buscar patrones de comportamiento. Sería preferible mediante el algoritmo *Apriori*, por ser el más extendido y utilizado en este estudio, pero *BigData* no nos proporciona este algoritmo, por lo que tendremos que optar por otras alternativas.
- Un mecanismo de Clasificación de alumnos mediante árbol de decisión para poder predecir la nota que obtendrán.

En cualquiera de los dos estudios, se adaptarán las métricas utilizadas para conseguir los objetivos en función de los datos que tengamos disponibles o podamos obtener de *Moodle*. Por ejemplo, [15] utiliza métricas relacionadas con mensajes directos entre usuarios de *Mood-*

le. Dada nuestra experiencia como alumnos y profesores de la UVa, estas herramientas no se usan demasiado, por lo que serán sustituidas por otras referidas a elementos de *Moodle* que más se usen el campus de la UVa. Por otro lado, existen métricas que se podrían calcular, como el tiempo dedicado en la realización de cuestionarios (*quiz*), pero en la práctica se ha visto que estos valores pueden no ser muy reales (se han visto ejemplos de tiempos de realización de *quiz* mayores de un año), lo que implica que el modo de conteo de tiempo de esa herramienta puede dar lugar a problemas si no se utiliza correctamente, por lo que se ha preferido no incluirla en la aplicación de los algoritmos.

En cuanto a la adaptación de [10], los resultados aquí obtenidos van a servir de base para realizar un cálculo estadístico previo de la actividad en *Moodle* así como para añadir más métricas de influencia que podemos obtener dadas las posibilidades que nos da la información que se guarda de *Moodle*.

El desarrollo de la adaptación de estos dos artículos, se detalla en los siguientes capítulos. El resto de requisitos del trabajo surgen a medida que avanza su desarrollo y en función de la estrategia de trabajo planteada.

Capítulo 4

Diseño e implementación de la solución

En este capítulo se desarrolla la solución planteada para abordar el problema indicado en el Capítulo 3. En la Figura 80 se muestra el esquema con el procedimiento que se va a seguir en el diseño de la solución. En primer lugar, se extraerán los registros de acceso a *Moodle* de la correspondiente base de datos y se transferirán, mediante *Sqoop* a nuestra infraestructura *Big-Data Hadoop*. En ella se realizará el procesamiento correspondiente para conseguir unos resultados que se almacenarán en una base de datos externa, implementada con *ElasticSearch*, haciendo uso del conector *Hadoop – ElasticSearch*. *Kibana* se encargará de leer los datos de *ElasticSearch* para generar con ellos gráficos que puedan servir para extraer conclusiones de los análisis realizados.

Figura 80. Esquema del diseño de la solución

Este capítulo se organiza de la siguiente forma. El apartado 4.1 detalla cómo preparar el entorno de trabajo *Hadoop* que proporciona *Hortonworks* para comenzar a trabajar, así como la

base de datos *ElasticSearch* para implementar la parte de presentación. En el apartado 4.2 se describe cómo se puede utilizar la herramienta *Sqoop* para transferir datos hacia/desde *Hadoop*. A continuación, en el apartado 4.3 se realiza un breve repaso al uso de *Hive* y su lenguaje asociado *HiveQL*, centrado principalmente en el uso que se va a hacer de él.

Antes de comenzar a detallar el trabajo realizado, en el apartado 4.4 se presenta la estructuración del código fuente que se proporciona junto con este documento, para que sirva a modo de glosario de la nomenclatura usada para describir el trabajo.

Para seguir con el esquema de la Figura 80, en primer lugar se describe qué información tenemos en el origen de datos, la base de datos de *Moodle*, qué necesitamos de ella y cómo la vamos a transferir, todo ello en el apartado 4.5. En segundo lugar, sobre la información obtenida, se realiza el análisis *BigData*, desarrollado en el apartado 4.6. Los resultados de este análisis se transfieren a *ElasticSearch*, para generar unos resultados gráficos como los que se enumerarán en el apartado 4.7. Toda esta ejecución se automatizará por medio de *Oozie*, tal y como se ilustrará en el apartado 4.8.

4.1. Preparación del entorno de trabajo

4.1.1. Instalación de la *sandbox* de *Hortonworks*

El primer paso es la instalación de la *sandbox* de *Hortonworks*, para ello seguimos los pasos indicados en la guía de instalación, accesible en: http://hortonworks.com/hdp/downloads/ de donde también se pueden descargar las imágenes con las máquinas virtuales. En el momento de realización de este trabajo la versión más reciente era *HDP 2.3.0*. De ese enlace descargamos el servicio virtualizado correspondiente en función de qué plataforma de virtualización queremos usar (*VirtualBox* o *VMWare*), esta decisión es indiferente para el resto del trabajo. En nuestro caso, hemos optado por *VirtualBox* ya que es gratuito en su totalidad y nos ofrece mayores funcionalidades para gestionar nuestras máquinas virtuales (ya que tendremos más de una). Antes de seguir se debe comprobar que nuestro ordenador anfitrión cumple los siguientes requisitos:

- Sistema operativo de 64 *bits*.
- Al menos 8 GB de *RAM* disponibles para ser asignados a la máquina virtual.
- Virtualización habilitada en la *BIOS*.

Una vez en *VirtualBox* exportamos el servicio virtualizado descargado de la página de *Hortonworks* y configuramos la máquina virtual. En este punto lo más importante está en la memoria *RAM* asignada, se recomienda un mínimo de 8 GB de *RAM* asignados a la máquina virtual para poder ejecutar correctamente todos los servicios. Tras unos minutos de exportación, y antes de lanzar la máquina, debemos configurar el modo de red de la MV, seleccionado Modo Puente (*bridge*) para que esta MV quede conectada a la red en la que se encuentra la máquina real. Podría ser conveniente reinicializar la dirección *MAC* de la MV para que se le asigne otra distinta a la que configuró *Hortonworks* y evitar problemas de duplicados de *IPs*. Tras ello, lanzamos la instancia tras lo cual podremos ver una pantalla, como la mostrada en la Figura 81, donde se indica la dirección *IP* que se ha asignado a la *sandbox* instalada. De igual forma, nos indica que para acceder a la consola de la MV usaremos el usuario `root` con clave `hadoop`.

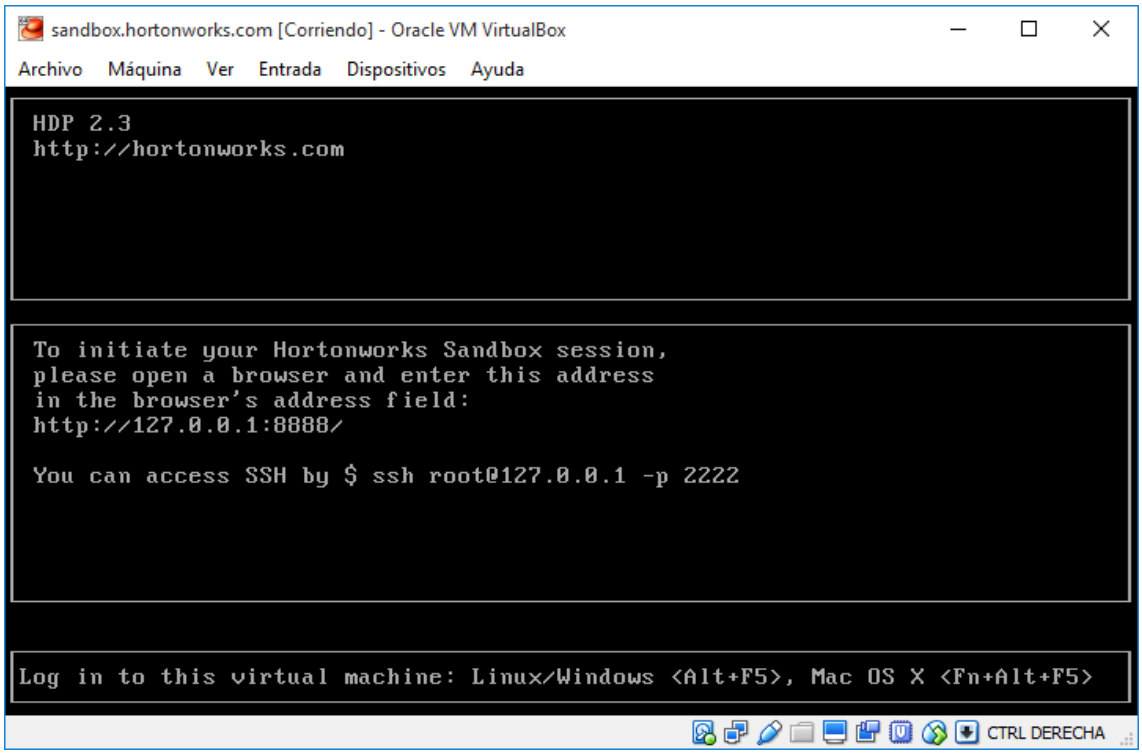


Figura 81. Página de bienvenida de la consola de la *sandbox*

4.1.2. Cuestiones generales

Tras haber instalado la *sandbox*, convendría realizar una serie de pasos antes de comenzar a trabajar con ella. Por ejemplo, en primer lugar, convendría actualizar el repositorio del gestor de paquetes. Como la *sandbox* está basada en una distribución *CentOS*, el gestor de paquetes es *YUM*, por lo que podemos actualizar su repositorio con la siguiente sentencia:

```
yum -y update
```

Sería conveniente configurar el idioma del teclado, para un eficiente uso de la consola, para ello, se debe modificar el fichero (por ejemplo mediante el editor `nano`, que debería ser instalado mediante `yum install nano`) `/etc/sysconfig/keyboard` y sustituir el idioma en las sentencias KEYTABLE y LAYOUT de tal manera que queden de la forma:

```
KEYTABLE="es"
LAYOUT="es"
```

Después, configuraremos fecha y hora de la máquina al huso horario de Madrid. Para ello:

- Instalamos el servidor de actualización de fecha por internet *NTP* (*Network Time Protocol*) a través de los paquetes `ntpdate` y `ntp`.

```
yum install ntpdate ntp
```

- Ejecutar el servicio *NTPD.*

```
service ntpd start
```

- Establecemos el huso horario de Madrid como característico de la máquina.

```
ln –sf /usr/share/zoneinfo/Europe/Madrid /etc/localtime
```

- Si vemos que la hora de la máquina está desincronizada podemos usar cualquiera de los dos siguientes comandos, para actualizarla:

```
service ntpdate restart
ntpdate –u ntp.ubuntu.com
```

Es importante tener en cuenta la actualización de la hora de la máquina, más si cabe si tenemos un *cluster* con varias máquinas, ya que *Hadoop* no funcionará correctamente si todas las máquinas no estás sincronizadas a la misma hora.

Otra fecha a cambiar es la hora de *Hue*, ya que será necesario para poder configurar correctamente la planificación horaria de los trabajos. Para ello, accedemos al fichero de configuración de *Hue,* `hue.ini` ubicado en `/etc/hue/conf.empty` para fijar:

```
time_zone=Europe/Madrid
```

En este fichero también es importante configurar la información del *JobTracker* (o *ResourceManager*) para que al lanzar trabajos se encuentre al gestor de *YARN*. Para ello, debemos garantizar la existencia de las siguientes líneas en este mismo fichero.

```
resourcemanager_host=sandbox.hortonworks.com
resourcemanager_port=8050
resourcemanager_rpc_url=http://sandbox.hortonworks.com:8050
resourcemanager_api_url=http://sandbox.hortonworks.com:8088
```

En caso de que se necesite instalar otra versión de *Hue*, se pueden seguir los pasos enumerados en [195].

La configuración de servicios se realiza desde el panel de mandos de *Ambari*, seleccionando el servicio correspondiente y después la pestaña de *Configs*. De entre todas las configuraciones que aquí se pueden hacer, en la realización de este trabajo han sido de utilidad, por servicio, las que se enumeran a continuación. Los ficheros de configuración utilizados se proporcionan junto con el código fuente del trabajo.

- ***MapReduce*** (ficheros `mapred-site.xml` y `mapred-env.sh`):
    - Tamaño del *container* para la parte *Map* de una operación *MapReduce*: *Map Memory* (`mapreduce.map.memory.mb`) a 1024 MB.
    - Tamaño del *container* para la parte *Reduce* de una operación *MapReduce*: *Reduce Memory* (`mapreduce.reduce.memory.mb`) a 1024 MB.

  - Tamaño del *container* del *ApplicationMaster* de una operación *MapReduce*: *AppMaster Memory* (`yarn.app.mapreduce.am.resource.mb`) a 1024 MB.
  - Cambios asociados en (con *Ambari*, se indican automáticamente):
    - `mapreduce.task.io.sort.mb`.
    - *MR Map Java Heap Size* (`mapreduce.map.java.opts`).
    - *MR Reduce Java Heap Size* (`mapreduce.reduce.java.opts`).
- ***YARN*** (ficheros `yarn-site.xml`, `yarn-env.sh` y `capacity-scheduler.xml`):
  - Memoria total dedicada a *YARN*: *Memory allocated for all YARN containers on a node* (`yarn.nodemanager.resource.memory-mb`).
  - Tamaño mínimo de *container*: *Minimum Container Size (Memory)* (`yarn.scheduler.minimum-allocation-mb`).
  - Tamaño máximo de *container*: *Maximum Container Size (Memory)* (`yarn.scheduler.maximum-allocation-mb`).
  - Número de núcleos virtuales asignados a *YARN*: *Number of virtual cores* (`yarn.nodemanager.resource.cpu-vcores`).
  - Número mínimo de núcleos virtuales asignados a un *container*: *Minimum Container Size (VCores)* (`yarn.scheduler.minimum-allocation-vcores`).
  - Número máximo de núcleos virtuales asignados a un *container*: *Maximum Container Size (VCores)* (`yarn.scheduler.maximum-allocation-vcores`)
  - Pila de recursos para el *ResourceManager*: *ResourceManager Java heap size* (`resourcemanager_heapspace`). Se dice que con 1024 MB es suficiente para la mayoría de trabajos *BigData* que se pueden definir
  - Definición de las colas de ejecución (apartado 4.8.2): *Capacity Scheduler* (`capacity-scheduler`). Por ejemplo, el contenido del fichero, `definición-colas.txt`.
- ***Hive*** (ficheros `hive-site.xml` y `hive-env.sh`):
  - Tamaño del *container* para la ejecución de trabajos *Tez*: *Tez container size* (`hive.tez.container.size`).
  - `hive.tez.java.opts` con un tamaño máximo (`-Xmx`) al 80 % del container *Tez*.
  - `hive.txn.manager = org.apache.hadoop.hive.ql.lockmgr.DbTxnManager` para activar la funcionalidad de transacciones que permite realizar operaciones `DELETE` y `UPDATE`.
  - `hive.enforce.bucketing = true` para poder realizar *bucket* (apartado 4.3.6) de tablas.
  - `hive.exec.dynamic.partition = true` y `hive.exec.dynamic.partition.mode = nonstrict` para fijar un modo de

partición dinámico no estricto (apartado 4.3.6). El número máximo de particiones se define en las propiedades `hive.exec.max.dynamic.partitions` y `hive.exec.max.dynamic.partitions.pernode`

  - Más información para una configuración *Hive* eficiente en términos de ejecución se puede encontrar en [196].

- En ***Oozie*** existe un parámetro con el que se indica el tamaño máximo que puede tener una salida resultado de un trabajo *Oozie*. Puede ocurrir que al lanzar algún trabajo, resulte un error relacionado con este criterio. Se puede aumentar este valor en *Oozie* => *Configs* => *Cuestom oozie-site* y añadir una nueva propiedad de nombre `oozie.action.max.output.data` y el valor del tamaño máximo expresado en bytes. Por defecto tiene un valor de 2048 bytes = 2 KB.

4.1.3. *URLs* de interés

En la Tabla 11 se indican todas las *URLs* de interés para el acceso a los distintos servicios, así como la clave de acceso, en caso de que sea necesaria.

URL	Descripción	Usuario	Contraseña
http://sandbox:8888	Página de bienvenida a la *sandbox* que nos proporciona acceso directo a los principales servicios		
http://sandbox:8080	Página de acceso al panel de mandos de *Ambari*	`admin`	`admin`
http://sandbox:8000	Página de acceso a la interfaz *Hue*	`hue`	`1111`
http://sandbox:4200	Cliente *web SSH* que se conecta a la *sandbox* para lanzar una consola remota alternativa a la MV	`root`	`hadoop`
http://sandbox:50070/dfshealth.html#tab-overview	Información sobre el estado del sistema de ficheros		
http://sandbox:50070/explorer.html#/	Acceso al sistema de ficheros *HDFS* para poder navegar por él y acceder a los ficheros (previa descarga)		
http://sandbox:11000/oozie/	Acceso a la consola de *Oozie* para monitorizar los trabajos lanzados		
http://sandbox:19888/jobhistory	*Job History*: monitorización de trabajos *MapReduce* lanzados		
http://sandbox:8088/cluster	Página principal del		

	ResourceManager para monitorizar las aplicaciones lanzadas
http://sandbox:18080/	Historial de trabajos *Spark*

Tabla 11. *URLs* de interés en *Hadoop*

4.1.4. Configuración del *cluster*

Una vez configurados los aspectos básicos de la *sandbox*, podemos configurar un *cluster*. Aunque no va a ser la infraestructura utilizada en este trabajo, se presenta aquí la forma de realizarlo.

Los nodos esclavos a añadir a la infraestructura deben ser configurados, antes de ser agregados al *cluster,* como nodos *Hadoop*. En concreto, se debe instalar un *SO* y dar ciertas configuraciones básicas. Una buena estrategia podría ser, crear un nodo "base" con toda la configuración necesaria y después clonar esa máquina para implementar los nodos esclavos, para que fácilmente se pueda en un futuro añadir nuevos nodos al *cluster* sin mayor complicación.

El nodo esclavo debe tener instalado un *SO CentOS*, por ejemplo, *CentOS 6.7*. Se puede descargar la imagen del *SO* de cualquiera de los *mirrors* indicados en http://isoredirect.centos.org/centos/6/isos/x86_64/. Durante la instalación de la imagen correspondiente, sería conveniente:

- Escoger un usuario `root` con contraseña `hadoop` para guardar similitud con la *sandbox* y no tener muchas correspondencias usuario – contraseña diferentes en cada MV.
- Crear un usuario denominado `hadoop` con contraseña `hadoop` para poder acceder a la interfaz gráfica de usuario.
- A la hora de decidir el tipo de máquina a instalar, escoger *Máquina para el desarrollo de software* (*Software Development Workstation*) para que se instalen los paquetes que un desarrollador de *software* necesitaría para desempeñar su trabajo, según el criterio del desarrollador de la imagen.
- Dar un nombre genérico a la máquina, ya que luego se cambiaría para cada nodo, por ejemplo, `nodobase.hortonworks.com`.

Una vez completada la instalación se configura el modo de red de la MV en puente y seguimos los siguientes pasos para dejar plenamente operativa la MV para poder formar parte del *cluster Hadoop* (todo desde el usuario `root`).

- Activar la interfaz de red donde se ha configurado el modo puente, para que se inicie con el arranque de la máquina y se le proporcione una dirección IP. Para ello, modificamos el fichero `/etc/sysconfig/network-script/ifcfg-ethX`, donde `X` es el número de interfaz, normalmente 0, para que contenga la línea:

```
ONBOOT=yes
```

- Deshabilitar IPv6. Añadiendo al fichero `/etc/sysctl.conf` las líneas:

```
net.ipv6.conf.all.disable_ipv6=1
net.ipv6.conf.default.disable_ipv6=1
```

Y ejecutando los comandos

```
sysctl –w net.ipv6.conf.all.disable_ipv6=1
sysctl –w net.ipv6.conf.default.disable_ipv6=1
```

- Deshabilitar cortafuegos, deteniendo los servicios e impidiendo que se inicien con el arranque del sistema:

```
service iptables save
service iptables stop
chkconfig iptables off
service ip6tables save
service ip6tables stop
chkconfig ip6tables off
service libvirtd stop
chkconfig libvirtd off
```

- Deshabilitar *THP* ejecutando los siguientes comandos:

```
echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
```

y modificando el fichero `/etc/rc.local` añadiendo las siguientes líneas:

```
if test -f /sys/kernel/mm/redhat_transparent_hugepage/enabled; then
  echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
                                fi

if test -f /sys/kernel/mm/redhat_transparent_hugepage/defrag; then
  echo never > /sys/kernel/mm/redhat_transparent_hugepage/defrag
                                fi
```

- Habilitar *NTPD* para actualizar automáticamente por red la fecha y hora de la máquina:

```
service ntpd start
chkconfig ntpd on
```

Tras esta configuración, el siguiente paso es generar un par de claves *SSH*, necesarias para el registro de los nodos en la *sandbox*. Para generar las claves se ejecuta el comando:

```
ssh-keygen
```

Con ello se generan un par de claves en el directorio `/root/.ssh/`, una clave pública `id_rsa.pub` y otra privada `id_rsa`. El contenido de la clave pública se debe copiar al fichero `authorized_keys` del mismo directorio, por ejemplo, ejecutando el comando:

```
cat /root/.ssh/id_rsa.pub > /root/.ssh/authorized_keys
```

La clave privada deberá ser trasladada a la *sandbox*. Si generamos este nodo base y lo clonamos para generar el resto de nodos esclavos, todos tendrán las mismas claves privada/pública, lo cual facilitará la configuración del *cluster Hadoop*.

Tras estos pasos podemos clonar la máquina base para generar los nodos esclavos. Sobre ellos se deberá hacer unos mínimos cambios:

- Antes de iniciar las máquinas, reinicializar su dirección *MAC* (si no se tuvo opción de hacerlo al clonar la máquina original) para evitar que tengan la misma *MAC* que la máquina original y por lo tanto la misma *IP*.

- Cambiar el nombre (*hostname*) del nodo modificando el fichero `/etc/sysconfig/network` y el apartado `HOSTNAME` para que los nodos tengan nombres del tipo `nodoXX.hortonworks.com`.

- Registrar todas las direcciones *IP* del *cluster Hadoop* en el fichero `/etc/hosts`. Es decir, se deberá registrar la dirección *IP* de la *sandbox* y de todos y cada uno de los nodos esclavos del *cluster*, por ejemplo, si tenemos un *cluster* con tres nodos, deberá ser de la forma:

```
ip_sandbox          sandbox.hortonworks.com
 ip_nodo01           nodo01.hortonworks.com
 ip_nodo02           nodo02.hortonworks.com
```

  Este registro también debe hacerse en la *sandbox*.

Con todos estos pasos ya se han configurado los nodos esclavos. El siguiente paso es añadirlos al *cluster Hadoop*. Para ello, desde la *sandbox*, accedemos a la interfaz de *Ambari*, mediante la *URL* http://sandbox:8080 y en ella al apartado de *Host*. En él aparece una lista con los *host* instalados en el *cluster*, inicialmente solo está la *sandbox*. Mediante la opción *Actions => Add New Hosts* accedemos a un asistente de instalación de nuevos *hosts* en el *cluster*. En la primera página del asistente (Figura 82), debemos seleccionar los nodos que queremos añadir mediante su nombre de *host* (*hostname*). Este nombre debe ser resoluble a una dirección *IP* por medio del fichero `/etc/hosts`, como se indicó anteriormente. En esta misma página debemos especificar la clave privada *SSH* que generamos previamente. En caso de que cada máquina tuviera una clave distinta, deberíamos realizar cada proceso por separado, porque el asistente no permite añadir simultáneamente nodos con distinta clave *SSH*. Podemos escribir la clave en el formulario correspondiente o indicar el fichero en que se guarda. Es preferible esta segunda opción ya que al escribir o copiar/pegar puede haber inconsistencias con los finales de línea que provoquen que la clave no sea correcta. Una alternativa por la que optar es transferir la clave privada desde cualquiera de los nodos esclavos a la máquina anfitriona por medio de *SFTP*, así no estamos manipulando el contenido de la clave y esta seguirá siendo válida. Después, desde el asistente, se añade la clave. Se debe tener en cuenta que todos los asistentes de subida de ficheros desde las interfaces *web* se lanzan sobre el sistema operativo anfitrión, por lo que el fichero que queramos subir deberá estar en el *SO* anfitrión y no en la

máquina virtual. Del resto de páginas del asistente, nos interesa detenernos en la página *Assign Slaves and Clients* dónde seleccionaremos que características queremos instalar en cada nodo. Como puede verse en la Figura 83, para cada uno de los nodos a instalar, podremos seleccionar qué servicios queremos instalar en él. En un caso genérico, se seleccionará *DataNode* (para que este nodo forme parte de *HDFS* y en él se almacenen datos), *NodeManager* (para que pueda usarse como nodo esclavo en YARN) y *Client* (para que se instalen los clientes de todos los servicios). El resto de posibilidades no se usarán en este trabajo y por ello no se añadirán. En caso de necesitarse en un futuro, se pueden agregar estando ya el nodo funcionando en el *cluster*, por lo que no hay ningún problema. En el resto de páginas se seleccionan los valores por defecto y se lanza la instalación. Tras ello, ya tendremos disponibles nuestros nuevos nodos en el *cluster*.

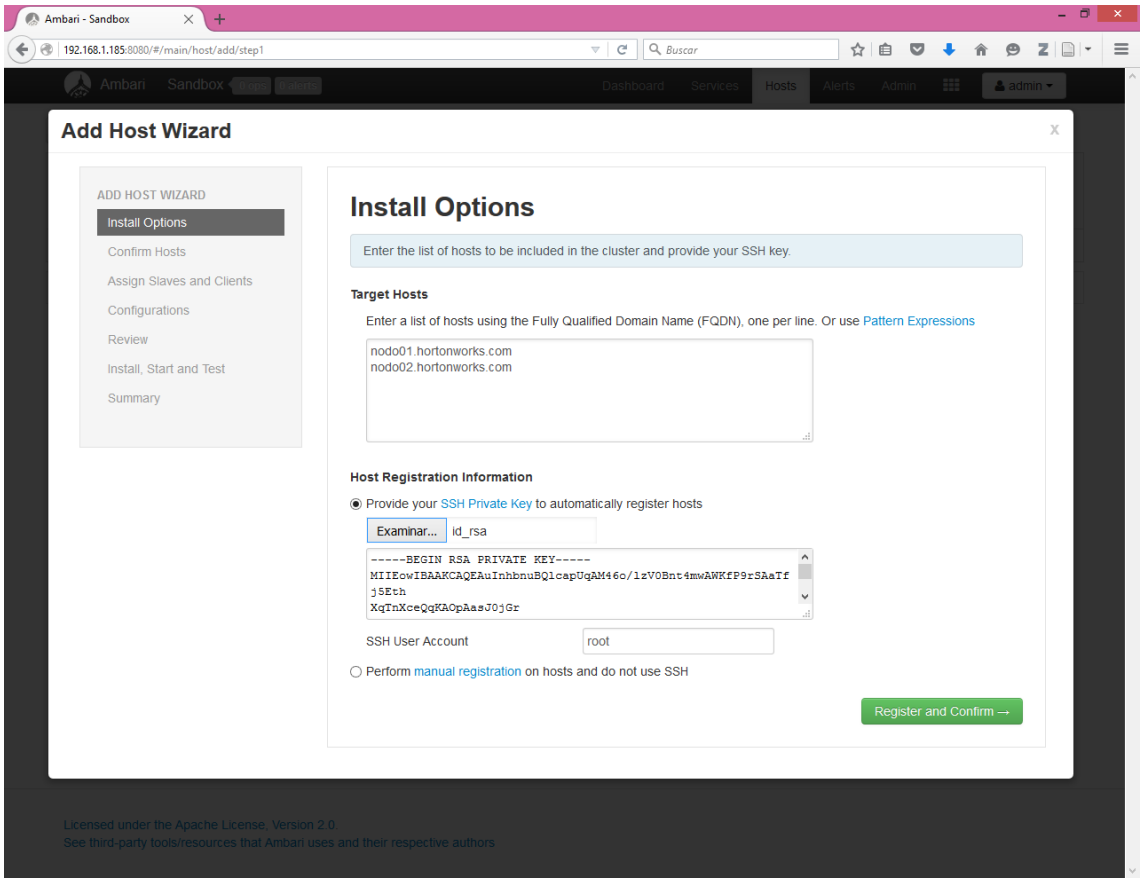


Figura 82. Asistente de adición de un nuevo host al *cluster* I

En este punto, se debe tener en cuenta una serie de cuestiones para que todo funcione correctamente.

- Tras la instalación, los diferentes servicios y clientes se inician, pero si el nodo esclavo se apaga, al reiniciarse deberán encenderse manualmente los servicios desde la interfaz de *Ambari* (*Hosts => nodoXX => Servicio => Start*).

- Tras reiniciar la *sandbox* se pierde todo contenido del fichero `/etc/hosts` por lo que las correspondencias nombre de *host* – IP de los nodos esclavos deberá ser otra vez configuradas.

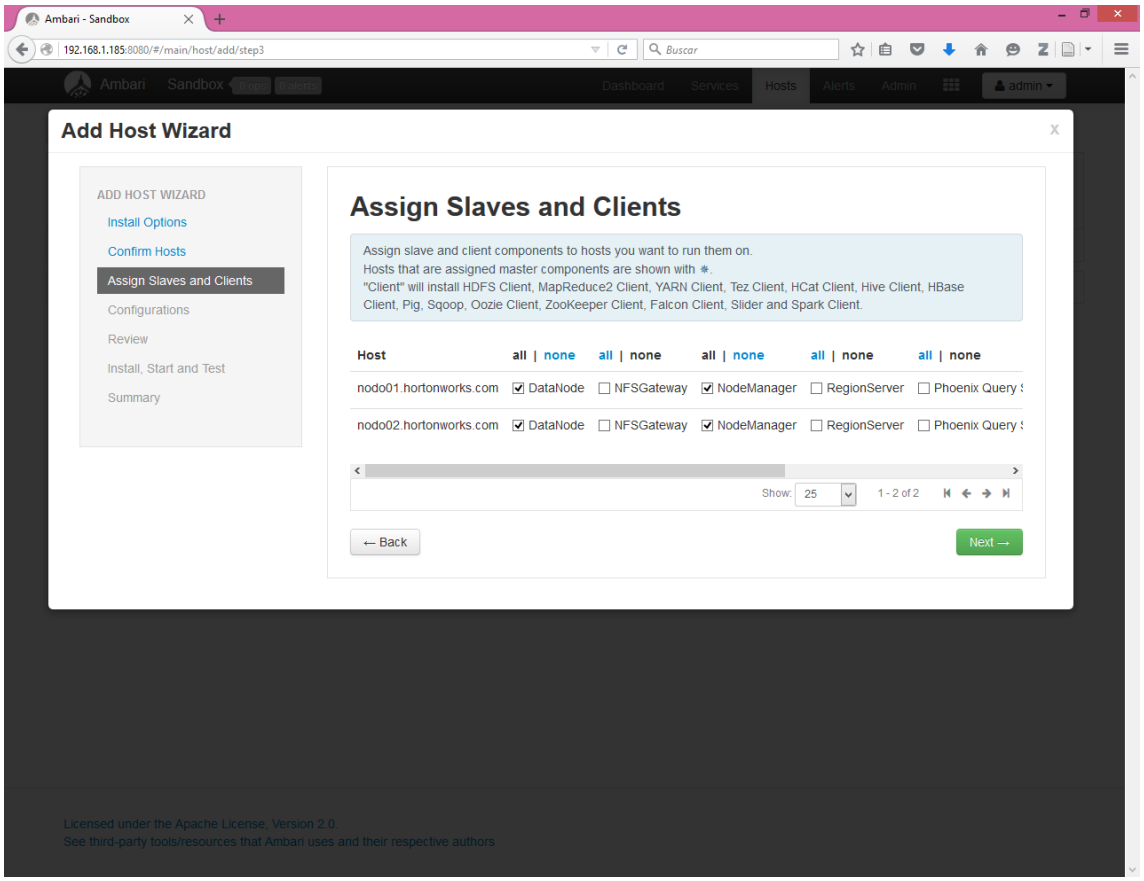


Figura 83. Asistente de adición de un nuevo host al *cluster* II

- En caso de que la dirección *IP* de algunas de las máquinas cambie, tan solo hay que cambiar las correspondencias en el fichero `/etc/hosts`.
- En caso de eliminar un *host* del *cluster* no se elimina de él la configuración *Hadoop* por lo que se puede volver a añadir aprovechando la instalación anterior.
- Si se elimina un *host* del *cluster* puede haber pérdida de datos si este era un *DataNode*.

Para la realización de este trabajo, no se ha configurado ningún *cluster* de nodos. Se ha utilizado directamente la *sandbox* como único elemento maestro – esclavo. Esta máquina virtual se levanta sobre un *SO Windows* 10 con 500 GB de disco duro y 32 GB de memoria *RAM*. De los 500 GB de disco duro, 150 GB están destinados a sistema operativo y los restantes a implementar un volumen de datos. Es en este segundo volumen donde se almacenarán las máquinas virtuales. La máquina virtual está configurada para tener un disco duro dinámico cuyo tamaño inicial (alrededor de 50 GB) se ha ampliado a un total de 300 GB, por medio de las herramientas de *Virtual Box*, y una memoria RAM asignada de 25 GB.

4.1.5. Uso básico de los servicios

Una vez configurado el *cluster*, ya podemos empezar a trabajar con él. En este apartado se va a presentar cómo deben configurarse los servicios para poder empezar a usarlos (configuraciones que se ha notado que faltan en la *sandbox* al ejecutar ejemplos básicos), así como algún comando básico para su uso. Recordar, que en la sección 2.2.3.2.9 se presentó un mini-tutorial sobre las opciones que nos proporcionan *Ambari* y *Hue* y que nos pueden ser de interés, como

aspectos y funcionalidades de los editores o navegación por *HDFS*. En este apartado, solo se indican ciertas configuraciones o pasos que hay que seguir para que todo funcione correctamente y que se han aprendido a base de prueba y error; con la excepción de lo referente a *Sqoop*, *Hive* y *Oozie*, que se detalla en los apartados 4.2, 4.3 y 4.8, repectivamente.

4.1.5.1. *HDFS*

Como se comentó en el Capítulo 2, se puede gestionar el sistema de ficheros *HDFS* mediante un navegador *web* y *Ambari* o *Hue*, o por medio de la consola. Esta última opción es la que se utiliza cuando necesitamos intercambiar datos entre *HDFS* y el sistema de ficheros de la MV. En la Tabla 12 se resumen los comandos más útiles para gestionar *HDFS* desde línea de comandos.

Comando	Significado
hadoop fs –put *path_local path_hdfs* **hadoop fs –copyFromLocal *path_local / path_hdfs***	Copia el contenido en el sistema de ficheros ubicado en *path_local* en la ruta ubicada en *path_hdfs* en *HDFS*
hadoop fs –get *path_hdfs path_local* **hadoop fs –copyToLocal *path_hdfs / path_local***	Obtiene el contenido de path_hdfs en *HDFS* y lo copia en la ruta *path_local* del sistema de ficheros local de la máquina
hadoop fs –mkdir *path_hdfs/Carpeta*	Crea el directorio *Carpeta* en la ruta path_hdfs en *HDFS*
hadoop fs –chown *user:user path_hdfs*	Cambia el dueño a *user* y el grupo a *user* del fichero o directorio ubicado en la ruta *path_hdfs* en *HDFS*. Si es un directorio se puede añadir la opción *–R* para hacerlo recursivo
hadoop fs –chmod [*modo*] *path_hdfs*	Cambia los permisos del fichero o directorio ubicado en *path_hdfs* aplicando el nuevo conjunto de permisos *modo*. Si es un directorio se puede añadir la opción *–R* para hacerlo recursivo
hadoop fs –mv path_hdfs_origen path_hdfs_destino	Mueve el fichero ubicado el *path_hdfs_origen* a *path_hdfs_destino*. Se puede usar para renombrar ficheros
hadoop fs –cp *path_hdfs_origen path_hdfs_destino*	Copia el fichero ubicado el *path_hdfs_origen* en *path_hdfs_destino*, siendo tanto el origen como el destino rutas dentro de *HDFS*
sudo –u *user* hadoop fs …	Añadiendo delante del comando sudo –u *user* se lanza la orden como el usuario *user*
*****	A las rutas se pueden añadir el comodín * para seleccionar más de un elemento

Tabla 12. Resumen de comandos básicos por terminal en *HDFS*

El acceso a *HDFS* mediante navegador *web* no tiene mayor complejidad y por ello no se detallará en este documento.

4.1.5.2. *Pig*

Para editar *scripts Pig* podemos usar *Ambari* o cualquier versión de *Hue*. Como ya se comentó en el Capítulo 2, la versión 3.8 de *Hue* no presenta la opción de incluir parámetros a la ejecución (o al menos no ha sido encontrada) por lo que esto puede restringir el uso de *Pig*, es por ello, que quizá sea mucho más adecuado usar *Ambari* o *Hue* 2.6. En cualquiera de los dos casos se ofrece al usuario unas plantillas con sentencias básicas a rellenar con los datos concretos del *script*. Hay que tener cuidado porque se ha advertido que hay errores en algunas de ellas, por ejemplo, para cargar datos usando *HCatalog* se genera una sentencia que contiene una llamada a una función `org.apache.hcatalog.pig.HCatLoader()` pero realmente debería ser `org.apache.hive.hcatalog.pig.HCatLoader()`; o de forma análoga para la función recíproca `HCatStorer()`. Por lo demás, no se requiere ninguna configuración especial.

4.1.5.3. *Spark*

En cuanto a ***Spark*** estamos en la misma situación que con *Sqoop*, es preferible manejarlo mediante línea de comandos. Ni *Ambari*, ni *Hue* 2.6 tienen interfaz *web* para *Spark*, aunque sí las nuevas versiones de *Hue*. Esta última interfaz permite ejecutar sentencias *Hive*, *Pig*, *Scala* o *Python* sobre *Spark*.

Si manejamos *Spark* mediante línea de comandos, tenemos como primera opción utilizar *spark-shell*, una consola que permite escribir sentencias *Scala*. También tenemos la opción de escribir programas con trabajos *Spark* en *Scala*, *Java* o *Python*, generar con ellos un ejecutable y lanzar este último. Si queremos generar trabajos con *Spark* que luego se puedan automatizar, esta última es la única alternativa, ya que tanto *spark-shell* como la interfaz nueva de *Hue* solo permiten escribir trabajos y ejecutarlos, pero manualmente.

El proceso típico de generación de un trabajo *Spark* comenzaría escribiendo un programa, por ejemplo en *Java*, usando funciones de *Spark*. A continuación se generaría un ejecutable *JAR* con este programa y se mandaría ejecutar mediante una línea de la forma:

```
spark-submit –class “clase main” –master tipoMaster path/a/ejecutable.jar
```

Donde deberemos especificar la clase principal (*main*) del ejecutable, el tipo de *Master* y modo, en caso de ser necesario (`local`, `local [k]`, `local [*]`, `yarn-client`, `yarn-cluster`, ver sección 2.2.3.2.4) y la ruta donde se encuentra, en el sistema de ficheros de la MV, el ejecutable con el programa *Java*. Si nuestro ejecutable está en *HDFS* y no en el sistema de ficheros de la MV, se deberá especificar de la forma:

```
“hdfs://sandbox:8020/ruta/a/ejecutable.jar”
```

Pero, solo se pueden utilizar ficheros almacenados en *HDFS* si se selecciona modo *yarn-cluster*. En caso de ser necesarios argumentos de entrada, estos se pondrán a continuación de la ruta, separados por espacios.

Por otro lado, si queremos usar la interfaz *Spark* de *Hue* 3.8 para lanzar sentencias, necesitamos tener instalado un servidor *Livy Spark.*

4.1.5.4. *Otros*

En cualquier momento, si queremos ver la lista de aplicaciones lanzadas sobre *YARN*, bien podemos acceder a la dirección http://sandbox:8088/cluster o ejecutar el comando:

```
yarn application –list
```

Mediante el acceso a la dirección previamente citada se obtiene mayor información así como acceso a los *logs*. Para acceder al *log* de una aplicación, deberemos conocer su identificador y usar el siguiente comando:

```
yarn logs -applicationId id_aplicación
```

Si queremos detener una aplicación:

```
yarn application –kill id_aplicación
```

4.1.6. Instalación de *ElasticSearch* y *Kibana*

La última etapa de nuestro análisis será la presentación de los resultados obtenidos con *Hadoop*. Previamente se ha comentado que se ha optado por extraer los datos de *Hadoop* a una base de datos externa, implementada con *ElasticSearch* y después representar esos datos con *Kibana*. En este apartado se explica cómo instalar *ElasticSearch* y *Kibana*.

En este trabajo se ha optado por implementar estas dos funcionalidades en una máquina distinta a cualquiera de las empleadas en la infraestructura *BigData Hadoop*. Principalmente para aislar ambos módulos y evitar que los problemas de uno afecten al otro. Además, se ha apreciado que en *Hadoop* hay instalada una versión de *ElasticSearch* antigua y que afectaría a lo que queremos configurar en este apartado, ya que no se ha encontrado forma de desinstalar y además no sabemos si puede afectar a otros componentes. De esta forma, la infraestructura de nuestro trabajo será como se muestra en la Figura 84, dentro de un entorno real *Windows* 10 tendremos dos máquinas virtuales con *CentOS*, una de ellas forma la infraestructura *Hadoop* (sandbox) y la segunda implementa la capa de presentación con *ElasticSearch* y *Kibana*.

Tanto *ElasticSearch* como *Kibana* son proporcionados por el mismo fabricante y podemos encontrar sus instaladores en la página de *elastic* [197].

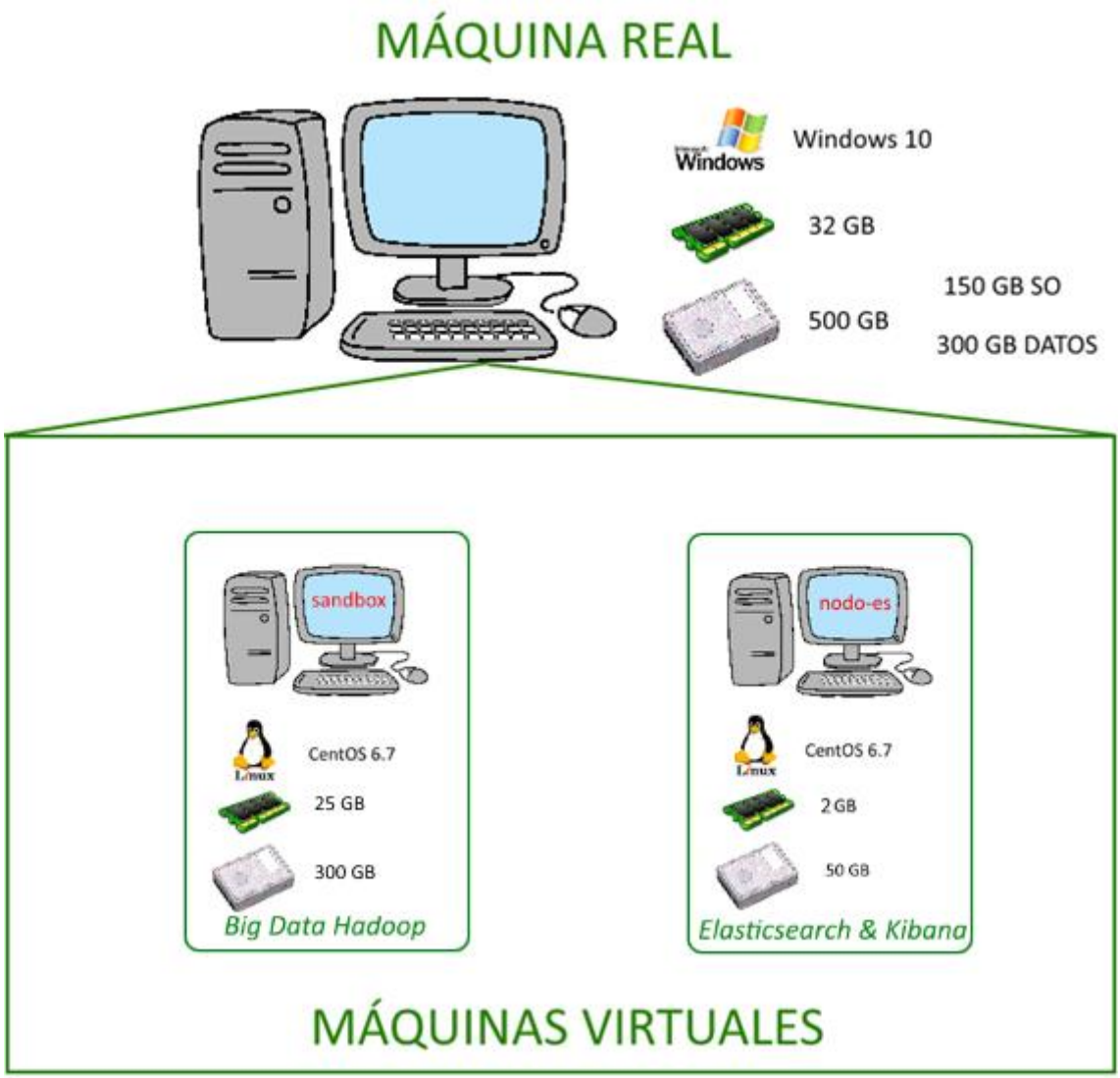


Figura 84. Esquema de infraestructura implementada con capa de presentación

Si empezamos instalando *ElasticSearch*, podemos escoger una forma de instalación mediante fichero comprimido *TAR-GZ* o en formato *RMP* para instalarlo con el gestor de paquetes. En nuestro caso hemos escogido la primera opción, ya que en caso que necesitemos desinstalar el *software* por algún problema de instalación o funcionamiento, no hay más que borrar un directorio. Descargamos la última versión de *ElasticSearch* de la página de *elastic*, por ejemplo de la forma:

```
wget
https://download.elasticsearch.org/elasticsearch/release/org/elasticsearch/di
stribution/tar/elasticsearch/2.0.0/elasticsearch-2.0.0.tar.gz
```

Y descomprimimos el fichero resultante:

```
tar –xvf  elasticsearch-2.0.0.tar.gz
```

Si se opta por la instalación mediante el gestor de paquetes y el fichero *RPM*, una vez descargado este:

```
wget https://download.elastic.co/elasticsearch/elasticsearch/elasticsearch-
1.7.3.tar.gz
```

Se instalaría de la forma

```
rpm –ivh  elasticsearch-2.0.0.rpm
```

En cualquiera de los dos casos, el servicio se configura en el fichero `elasticsearch.yml`, donde especificaremos:

- El nombre del *cluster ElasticSearch* en `cluster.name`. En nuestro caso, `eduvalab-es`.
- Nombre de este nodo en el *cluster* en `node.name`. En nuestro caso, `nodo-es`.
- Si este nodo es maestro o esclavo. En nuestro caso será ambas cosas ya que nuestro *cluster ElasticSearch* estará compuesto por un único nodo. De esta forma, deberemos indicar: `node.master:true` y `node.data:true`.
- Puerto de escucha en `http.port:9200`, y el puerto de operación `transport.tcp.port:9300`.
- Las propiedades `index.number_of_shard` y `index.number_of_replicas`, nos permiten fijar el número de *shards* y el factor de replicación, respectivamente. En nuestro caso, este último parámetro no tienen ningún sentido, ya que no tenemos otros nodos por donde distribuir las réplicas, pero para que la configuración sea correcta deberemos darle un valor de 0, ya que no puede generar réplicas al no tener por donde repartirlas.

En caso de querer implementar un *cluster ElasticSearch*, toda esta instalación debe hacerse en cada uno de los nodos. La única parte diferente sería la definición de si el nodo es maestro o de datos, o ninguno de los dos (balanceador). En el código fuente del trabajo se proporciona el fichero de configuración utilizado, por si puede servir de ejemplo.

Una vez configurado, lanzaremos *ElasticSearch* accediendo al directorio `bin` dentro del directorio donde descomprimimos la descarga del *software* y lanzando el programa `elasticserach`:

```
./elasticsearch
```

Si instalamos el servicio por medio del gestor de paquetes, lo gestionaremos por medio del comando `service`:

```
service elasticsearch start
```

Tras ello, podremos acceder a nuestra nueva base de datos mediante un navegador *web* y la dirección http://ip-nodo-es:9200. De entre las distintas alternativas de interfaz para usar *ElasticSearch* hemos optado por usar *Plugin Head*, el cual se puede instalar con el siguiente comando:

```
…/bin/plugin –install mobz/elasticsearch-head
```

Con esta interfaz podremos lanzar consultas sobre *ElasticSearch* por medio de su *API REST*, consultar el contenido de cada índice, ver cómo están repartidos los índices por los distintos nodos, qué nodo están activos, etc.

Algunas direcciones de interés se muestran en la Tabla 13:

Dirección	Descripción
http://ip-nodo-es:9200/_cat/health?v	
http://ip-nodo-es:9200/_cat/nodes?V	
http://ip-nodo-es:9200/_plugin/head	Página de inicio de *Plugin Head*
http://ip-nodo-es:9200/_cat/indices	Información de los índices que contiene

Tabla 13. Dirección de interés en *ElasticSearch*

En nuestro caso, la información la vamos a añadir a partir de *Hive* con sentencias *HiveQL*, para ello el procedimiento es cómo sigue.

En primer lugar, deberemos crear un índice definiendo la estructura de su contenido, como si estuviéramos indicando los atributos de las columnas de una tabla en una base de datos *SQL*. Como *ElasticSearch* se maneja por medio de una *API REST* podremos usar el comando *CURL* para enviar consultas. La creación del índice se realiza con una operación *PUT*. Por ejemplo:

```
curl -v -XPUT http://ip-nodo-es:9200/idx_foo
curl -v -XPUT ‘http://ip-nodo-
es:9200/idx_foo/_mapping/analysis?ignore_conflicts=true’ -d ‘{
  "analysis" : {
    "properties" : {
      "analyserVersion" : {
        "type" : "string",
        "store" : "true",
        "index" : "not_analyzed"
      },
      "analysisTime" : {
        "type" : "date",
        "format" : "date_optional_time"
      },
      "documentIdentifier" : {
        "type" : "string",
        "store" : true,
        "index" : "not_analyzed"
      },
      "topics" : {
        "type" : "string"
      }
    }
  }
}’
```

Con la primera sentencia `curl`, estamos añadiendo un índice denominado `idx_foo`. La segunda sentencia `curl` va a generar, para este índice, una estructura de documento que se ha denominado `analysis` y caracterizada por los siguientes atributos:

- `analyserVersion` de tipo *string*.
- `analysisTime` de tipo *date* (*date_optional_time* especifica el formato concreto de fecha).
- `documentIdentifier` de tipo *string*.
- `topics` de tipo *string*.

Algunos de estos parámetros llevan asociada una propiedad `"index":"not analyzed"`. Con ella se está especificando que, en caso de que el *string* contenga más de una palabra, no las analice por separado, si no que considere el total de palabras como una única. Por ejemplo, podemos querer contar el número de veces que aparece cada campo *analyserVersion*, para ver cuantos documentos son de cada versión. Si el campo de versión se guarda de la forma "Versión X" donde X es el número de versión, sin especificar el parámetro anterior, se analizarían por separado el número de veces que aparece la palabra "Versión" y el número de veces que aparece cada valor que pueda tener X. Si indicamos este parámetro, se analizará el número de veces que aparece cada conjunto "Versión X" para los distintos valores de X.

En caso de necesitar añadir información a los índices por medio de la *API REST*, se realizaría por medio de una operación *POST* indicando el índice y el tipo de documento a añadir en él, por ejemplo, de la forma:

```
curl -v -XPOST http://ip-nodo-es:9200/_bulk -d '
{ "create": { "_index": "idx_foo", "_type": "analysis" } }
{ "analyserVersion": "0.0.1", "analysisTime": "2015-03-07T16:12:00Z", "docu-
mentIdentifier": "doc1", "topics": "business" }'
```

En nuestro caso, trataremos de hacer toda la carga de datos por medio de *Hive*. En primer lugar tendremos que crear una tabla que esté relacionada con el contenido de un índice *ElasticSearch*. Para ello, tendremos que descargar el conector *Hadoop – ElasticSearch* de la página de *elastic*. Este es una librería *JAR* que debe copiarse en la librería de *Hive* `/usr/hdp/{versión}/hive/lib`. Tras esto, la tabla asociada se crea con una sentencia del tipo:

```
CREATE EXTERNAL TABLE nombre_tabla (
  analyser_version STRING,
  analysis_time TIMESTAMP,
  doc_id STRING,
  topics STRING
)
STORED BY 'org.elasticsearch.hadoop.hive.EsStorageHandler'
TBLPROPERTIES('es.nodes'='ip-nodo-es:9200','es.resource' = 'idx-
foo/analysis','es.mapping.names' = 'doc_id:documentIdentifier, analy-
sis_time:analysisTime, analyser_version:analyserVersion');
```

En las propiedades de tabla:

- `es.nodes` nos permite indicar el nodo *ElasticSearch.*
- `es.resource` apunta al nombre del índice y el tipo de documento. La estructura de este tipo de documento deberá coincidir con la que se defina para la tabla.

- `es.mapping.names` nos permite "transformar" los nombres de los atributos del índice en otros que nos puedan ser más útiles o fáciles de manejar en *Hive*. En caso de no especificar esta propiedad, el nombre del atributo en la tabla *Hive* deberá ser el mismo que se haya dado al crear el tipo de documento en *ElasticSearch*. El orden de definición de la tabla y el índice no tiene por qué ser el mismo, ya que busca atributos con el mismo nombre, o su correspondencia por medio de esta propiedad.

Tras ejecutar estas sentencias, si el índice *ElasticSearch* cargado tiene algún dato, lo podremos visualizar lanzando una consulta `SELECT` a la nueva tabla. Por ejemplo:

```
SELECT * FROM nombre_tabla
```

Para añadir información al índice *ElasticSearch* se puede utilizar una sentencia `INSERT`, pero previamente se debe indicar a *Hive* que use una serie de conectores necesarios:

```
ADD jar /usr/hdp/{versión}/hive/lib/elasticsearch-hadoop-2.1.1.jar;
ADD jar /usr/hdp/{version}/hive/lib/commons-httpclient-3.0.1.jar;
INSERT INTO nombre_tabla VALUES ('0.0.1','2015-03-07 16:12:06',
'doc12','target');
```

El problema que se ha advertido en la práctica es que los editores *Hive* de *Apache Ambari* o *Hue* no entienden las sentencias `ADD JAR` y dan error. Si se lanza desde el editor por línea de comandos funcionan correctamente. También funcionan correctamente si se lanza, mediante *Oozie* un *script HiveQL*. De esta forma, cuando estemos probando la infraestructura o tengamos que generar manualmente los índices, lo tendremos que hacer con el editor *Hive* por línea de comandos. El hecho de que funcione correctamente cuando se lanza mediante *Oozie* nos sigue permitiendo automatizar la ejecución de los trabajos.

Más rápida y sencilla es la instalación de *Kibana*. En este caso descargaremos el *software* para la versión *Linux* 64 *Bits* como un fichero *TAR GZ*:

```
wget https://download.elastic.co/kibana/kibana/kibana-4.1.2-linux-x64.tar.gz
```

Y descomprimiremos el fichero resultante:

```
tar –xvf kibana-4.2.0-linux-x64.tar.gz
```

Solo tendremos que configurar el fichero `kibana.yml`, ubicado en el directorio `config`, para comprobar la existencia de la línea:

```
elasticsearch.url: "http://ip-nodo-es:9200"
```

Donde indicamos cual es la dirección de acceso al *cluster ElasticSearch*. En el código fuente del trabajo puede consultarse el fichero de configuración *Kibana* utilizado.

Para lanzar este servicio, simplemente accedemos al directorio `bin` y lanzamos el ejecutable *kibana*:

```
./kibana
```

El acceso a *Kibana* es por medio de una navegador *web* y la dirección http://ip-nodo-es:5601. Usamos `ip-nodo-es` como dirección IP de nodo, ya que instalaremos *ElasticSearch* y *Kibana* en la misma máquina. Si estuvieran en diferentes nodos deberíamos indicar la *IP* del nodo donde hemos instalado y ejecutado *Kibana*. Al entrar por primera vez, nos pide que configuremos el/los índice/s sobre el que vamos a basar nuestras búsquedas.

En este trabajo se han utilizado las versiones *ElasticSearch* 1.7.3 y *Kibana* 4.1.2 ya que estas fueron las versiones utilizadas durante la etapa de pruebas de *software*. Antes de comenzar a trabajar con estos módulos aparecieron nuevas versiones, pero optamos por seguir con las anteriores ya que los ficheros de configuración eran más completos y simplemente teníamos que seleccionar las opciones que nos hicieran falta. En las nuevas versiones, la configuración inicial es mínima y se requiere conocer la sintaxis concreta de las opciones y añadirlas manualmente.

4.2. Transferencia de datos mediante *Sqoop*

Sqoop se maneja únicamente mediante línea de comandos, no existe interfaz *web* para manejarlo. Las versiones nuevas de *Hue* (no presentes en la *sandbox* de *Hortonworks*) proporcionan una interfaz para acceder a *Sqoop*, pero hay que tener cuidado porque no se refiere al *Sqoop* instalado en *Hortonworks*. En nuestra *sandbox* tenemos instalado el cliente *Sqoop1*, mientras que esa interfaz hace uso del servidor *Sqoop2* no presente en *Hortonworks* y por tanto no nos es de utilidad. Aunque se maneje por línea de comandos, la complejidad no va a ser grande. De entre todas las utilidades que proporciona analizaremos tres: la importación "normal" de datos, importación incremental y, aunque en nuestro trabajo no la utilizaremos, la exportación. La información que aquí se presenta está orientada al uso que vamos a hacer de *Sqoop*, existen otras muchas más herramientas *Sqoop* pero en nuestro trabajo no han sido necesarias, se pueden consultar en [175]. El funcionamiento interno de estas herramientas ya fue presentado en el apartado 2.2.3.2.7.

4.2.1. Importación simple

La **importación** que en este apartado se trata se ha denominado "normal" en el sentido de que no impone ninguna condición, en contra de lo que hace la importación incremental. Dentro de esta categoría, podemos distinguir dos tipos de importación: individual y de todas las tablas. En nuestro caso, nos interesa la importación desde un origen externo a *Hive*, por lo que nos centramos en las opciones posibles para manejar este caso.

Si empezamos por la **importación individual**, esta se refiere a la transferencia de una única tabla. Se realiza por medio del comando `import` genérico de *Sqoop* el cual se convierte en una carga sobre *Hive* si le añadimos la opción `--hive-import`. La sentencia utilizada para la importación podría ser de la forma:

```
sqoop import --connect jdbc:{tipoBD}://{ip_BD}:{port_BD}/{BD}  --table
{nombre_tabla} --username {usuario} --password {constraseña} --hive-import
                    --hive-table {name} --hive-overwrite
```

Donde, las opciones indicadas son las que de mayor utilidad pueden servir para este trabajo:

- Opciones genérica de `sqoop import`:
    - En {`tipoDB`} deberemos indicar el tipo de nuestra base de datos externa, como `postgresql`, `mysql`, etc.
    - En {`ip_DB`} deberemos indicar la *IP* de la máquina que aloja nuestra base de datos externa.
    - {`port_DB`} es el puerto de acceso en esa máquina a la base de datos.
    - {`BD`} es el nombre de la base de datos donde se encuentra la tabla a importar, en la máquina externa.
    - En {`nombre_tabla`} indicaremos el nombre de la tabla que queremos importar.
    - Con la pareja `usuario` – `contraseña` indicamos un usuario que pueda acceder a esa base de datos, por lo menos con privilegios de lectura.
- Opciones propias de la importación a *Hive*:
    - `--hive-import` indica a la herramienta que debe importar los datos a una tabla *Hive* cuyo nombre viene dado en `--hive-table`. Este último parámetro es opcional, si no se indica, se coge el nombre de la tabla importada.
    - Si especificamos `--create-hive-table` le indicamos que debe crear la tabla, por lo que si esta existe, dé el trabajo como fallido.
    - La opción `--hive-overwrite` nos permite sobrescribir el contenido de la tabla, es decir, borrar sus datos y modificarlos por los nuevos, pero solo en el caso en que estos tengan la misma estructura, ya que la tabla sigue siendo la misma. Si no se especifica, se añaden los nuevos datos a los anteriores.
    - Se pueden consultar más opciones en el manual de *Sqoop* [175]

La otra opción es **importar todas las tablas** presentes en una base de datos, para lo cual se requiere que todas ellas tengan un campo de clave única. En este caso, la sentencia sería similar a la anterior, sustituyendo `import` por `import-all-tables` y eliminado la opción `--table` ya que en este caso no debemos seleccionar una tabla de origen.

4.2.2. Importación incremental

La **importación incremental** nos permitirá importar solo cierta información de las tablas externas, principalmente para transferir solo los nuevos registros de las tablas en función de los que ya hemos importado previamente. Se pueden distinguir dos tipos de importación incremental:

- **De adición** (*append*): se utiliza cuando en la tabla externa tenemos garantizada la existencia de una columna cuyo valor no se repite entre registros. Cada vez que se añade

un nuevo registro, se incrementa este campo, por lo que se basa en traer a *Hadoop* aquellos registros cuyo valor para esta columna es mayor al último transferido, indicado en el comando. Para que la importación sea incremental en modo *APPEND* a cualquiera de las sentencias de importación anteriores habría que añadirles:

- `--incremental append`: escoger tipo de importación incremental.
- `--check-column {columna}`: escoger la columna con el campo incrementado.
- `--last-value {valor}`: último valor transferido para ese campo.

- **En función de última fecha de modificación** (*lastmodified*): se utiliza cuando la tabla externa puede tener registros que se actualicen, ya que la importación condicionada al valor de un campo puede perderse estos cambios. En este caso, se necesita, que cada vez que se realice un cambio, se registre la fecha de modificación en algún otro campo. Traeremos aquellos registros cuya marca temporal sea mayor a un valor dado, por ejemplo, la marca temporal del último registro importado. Para que la importación sea incremental en modo *LASTMODIFIED* a cualquiera de las sentencias de importación anterior habría que añadirles:
  - `--incremental last_modified`: escoger tipo de importación incremental.
  - `--check-column {columna}`: escoger la columna con la marca temporal.
  - `--last-value {valor}`: último valor transferido para ese campo.

4.2.3. Exportación

Aunque en este trabajo no va a ser necesario, también existe un comando de exportación, similar al anterior de importación:

```
   sqoop export --connect jdbc:{tipoBD}://{ip_BD}:{port_BD}/{BD} --table
  {nombre_tabla} --username {usuario} --password {contraseña} --export-dir
{path} --input-fields-terminated-by {patrón1} --lines-terminated-by {patrón2}
```

Para que este comando funcione, previamente se debe haber creado una tabla, en la base de datos externa, con nombre `nombre_tabla` y cuyos campos concuerden con los datos que queremos exportar. El conjunto de datos a exportar se especifica en `--export-dir` indicando la ruta en *HDFS* donde se encuentran los datos, por ejemplo, si es una tabla en *Hive*, indicando la ruta hacia su almacenamiento en *HDFS*: `/apps/hive/warehouse/nombre_tabla_hive`. Con las opciones `--input-fields-terminated-by` y `--lines-terminated-by` se indica cómo están delimitados los datos en el fichero de texto que se está exportando para que al ser almacenados en la base de datos externa, se sepa cuándo termina un campo y comienza el siguiente.

Un error típico es que los datos que queramos exportar no estén delimitados, porque no hayamos definidos delimitadores cuando estuvimos manejando las tablas. Una forma de solucionarlo, sin tener que borrar todas las tablas y volver a comenzar es realizar pasos intermedios entre la tabla con los resultados y la exportación, en concreto, crear una tabla idéntica a la de los resultados, pero delimitada, por ejemplo de la forma:

```
CREATE TABLE nombre_aux(columnas…) ROW FORMAT DELIMITED FIELDS TERMINATED BY
','
```

Así le estamos indicando que cada campo de una fila se separa con comas (,) y ya podemos expresar en el comando de exportación en la opción `--input-fields-terminated ','`. Típicamente las líneas terminan con retorno de línea \n, a menos que se especifique lo contrario.

4.2.4. *Drivers* de conexión

Para que todo funcione correctamente necesitamos el *driver* de la base de datos externa que estemos utilizando, por ejemplo, si nuestra base de datos externa es *MySQL* necesitamos darle a *Sqoop* el driver *MySQL*. En concreto deberemos situarlo en el directorio, del sistema de ficheros de la máquina virtual, `/usr/hdp/{versión}/sqoop/lib`.

4.3. *Hive* y Lenguaje *HiveQL*

En este apartado trataremos con *Hive* y su lenguaje de programación *HiveQL*. Esta va a ser la base de nuestro trabajo, ya que aproximadamente el 90% de él estará realizado sobre *Hive*. A la hora de realizar el análisis *BigData* teníamos dos opciones: utilizar *Hive* o *Pig*. No suelen ser herramientas excluyentes, sino complementarías, dependiendo de lo que queramos hacer o de nuestra pericia con sus correspondientes lenguajes utilizaremos la una o la otra. También, se suele considerar que *Hive* es adecuado cuando se trabaja con datos estructurados (como es nuestro caso) y *Pig* con datos no estructurados. En el caso de este trabajo, no solo por esto parece más adecuado *Hive*, sino también, porque *HiveQL* es bastante parecido a *SQL* con alguna diferencia, y por lo tanto el tiempo de aprendizaje del nuevo lenguaje, ha sido menor.

Para completar este apartado, brevemente se comenta cómo se puede acceder a usar *Hive* y nociones generales sobre su uso mediante consultas en *HiveQL*. El resto del apartado se completa con resúmenes de la documentación de *HiveQL* relativos a las cuestiones más importantes a conocer y saber para poder realizar este trabajo.

4.3.1. Nociones generales sobre *Hive* y *HiveQL*

Para usar *Hive* tenemos la línea de comandos, *Ambari* o *Hue*. En cuanto al funcionamiento, recordar que cuando se crea una tabla en *Hive*, esta se almacena como un fichero en el directorio `/apps/hive/warehouse/nombre_BD/nombre_tabla`. Si no se especifica nada, el fichero se guarda en formato texto plano, pero nosotros queremos formato *ORC* (ver sección 2.2.3.2.2), por lo que al crear una tabla, deberemos indicar que use este formato:

```
CREATE TABLE nombre (columnas…) STORED AS ORC
```

En cuanto a los formatos de ficheros disponibles son varias las opciones. En principio, sobre *Hortonworks*, son posibles de usar el texto plano y *ORC* (sin tener que configurarlos), mientras que otras dos de las opciones más típicas, *Avro* y *Parquet* vienen instaladas en *Cloudera*. En la Figura 85 se puede ver una comparativa del tamaño que ocuparía la misma información, codi-

ficada según varios estándares. De entre todos los comparados, *ORC* (recomendado para *Hortonworks*) permite disminuir hasta en un 78% el tamaño de los datos, así como permitir que las sentencias *Hive* se puedan ejecutar más eficientemente. Esto último se ha comprobado en la práctica: las sentencias se ejecutan con más rapidez, los resultados se guardan con más rapidez (ocupando menos espacio) y se generan menos situaciones de falta de memoria (*heap space – out of memory*).

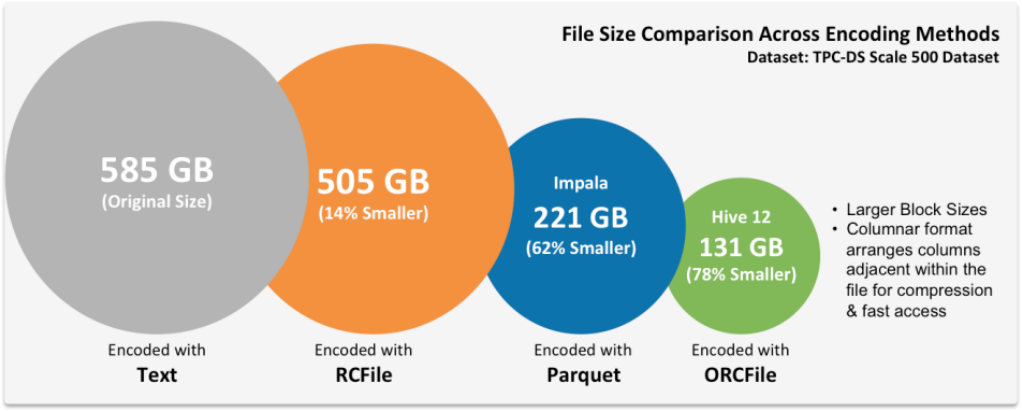


Figura 85. Comparativa entre distintos formato de ficheros en *Hive*

Una vez creada la tabla, tenemos distintas formas de cargar los datos, bien manualmente mediante sentencias *SQL* del tipo:

```
INSERT INTO nombre_tabla VALUES (lista1), (lista2)…
```

O aprovechándonos que *Hive* maneja los datos como ficheros, por ejemplo, de texto. Para utilizar esta segunda opción debemos tener en cuenta cómo hemos configurado, durante la creación de la tabla, que se almacenarán los datos (texto, *ORC*, etc.). Si por ejemplo hemos seleccionado texto, lo único que tenemos que hacer es coger un fichero que contenga los datos, delimitados según se definió en la creación de la tabla, y copiar este fichero al directorio que esta tabla tiene en el *warehouse* de *Hive*, y los datos ya estarán directamente cargados y se podrán consultar con el editor de consultas.

Una última alternativa también usa un fichero donde se encuentran todos los datos, pero en este caso usa sentencias *HiveQL* para cargar los datos, con la única condición de que ese fichero se encuentre en *HDFS* y se índice la ruta en el campo `path` del siguiente comando:

```
LOAD DATA INPATH path OVERWRITE INTO TABLE nombre_tabla
```

4.3.2. Tipos de datos

En *Hive*, se definen la mayor parte de los formatos de datos que se pueden usar en SQL. En la Tabla 14, se enumeran los tipos de datos definidos.

Tipos de datos aceptados en *HiveQL*	
Tipos numéricos	
TINYINT	SMALLINT
INT	BIGINT
FLOAT	DOUBLE
DECIMAL	
Tipos de fecha/tiempo	
TIMESTAMP	DATE
Tipos de cadena	
string	varchar
char	
Miscelánea	
BOOLEAN	BINARY
Tipos complejos	
ARRAYS	MAPS
STRUCTS	UNIONS

Tabla 14. Formato de datos aceptados en *HiveQL*

Se puede encontrar información respecto a ellos en [198].

4.3.3. Funciones *HiveQL* utilizadas

Son muchas las funciones que *HiveQL* proporciona para manejar los datos, algunas ya existentes en otros lenguajes *SQL* (aunque puede que con otro nombre). En la Tabla 15 se enumeran las utilizadas en este trabajo, junto con su descripción. Las funciones se dividen en tres categorías:

- ***UDF*** (*User Defined Functions*): para cada columna de la tabla, reciben un solo registro, pero pueden recibir varias columnas de un mismo registro. Con estos realiza un procesamiento por el que se devuelve un único resultado.
- ***UDAF*** (*Built-in Aggregated Functions*): para cada columna pueden recibir información de varios registros y devuelven un valor, por ejemplo, la función que calcula un máximo `max()` de entre un conjunto de valores, recibe una columna con todos los valores y devuelve un único valor, el máximo encontrado.
- ***UDTF*** (*Built-in Table-Generating Functions*): recibe una entrada simple y genera varias salidas, por ejemplo, la función `explode()`, dado un *array*, devuelve una columna compuesta por los elementos del *array*.

Funciones *HiveQL*		
Función	**Tipo devuelto**	**Descripción**
UDF		
Operadores relacionales		
A = B	BOOLEAN	Devuelve TRUE si A es igual a B, FALSE en cualquier otro caso
A != B	BOOLEAN	Devuelve TRUE si A no es igual a B, FALSE en cualquier

		otro caso
A < B	BOOLEAN	Devuelve TRUE si A es menor que B, FALSE en cualquier otro caso
A <= B	BOOLEAN	Devuelve TRUE si A es menor o igual que B, FALSE en cualquier otro caso
A > B	BOOLEAN	Devuelve TRUE si A es mayor que B, FALSE en cualquier otro caso
A >= B	BOOLEAN	Devuelve TRUE si A es mayor o igual que B, FALSE en cualquier otro caso
A [NOT] LIKE B	BOOLEAN	Devuelve TRUE si el *string* A [no] es parecido al *string* B, FALSE en caso contrario. La comparación se hace carácter a carácter.
Operadores aritméticos		
A + B	Numérico	Suma de A y B
A – B	Numérico	Resta de A y B
A * B	Numérico	Multiplicación de A y B
A / B	Numérico	División de A entre B
Operadores lógicos		
A IN (val1, val2,…)	BOOLEAN	Devuelve TRUE si A es igual a cualquiera de los valores `valN`
Conversión de tipos		
cast(expr as <type>)	`<type>`	Convierte el resultado de la expresión `expr` al formato `<type>`. Si la conversión no tiene éxito, devuelve `null`
Funciones con fechas		
unix_timestamp()	`bigint`	Obtiene la marca de tiempo *Unix* actual, en segundos
unix_timestamp(string date)	`bigint`	Convierte el *string* `date`, supuesto en formato YYYY-MM-dd HH:mm:ss (representando una fecha), a su marca de tiempo *Unix*, en segundos
unix_timestamp(string date, string pattern)	`bigint`	Convierte el *string* `date`, supuesto en formato especificado por `pattern` (representando una fecha), a su marca de tiempo *Unix*, en segundos
to_date(string timestamp)	`string`	Devuelve la parte de la marca de tiempo que representa el día, es decir, YYYY-MM-dd
date_add(string startdate,	`string`	Suma el número de días

int days)		days a la fecha inicial `startdate`
date_sub(string startdate, int days)	`string`	Resta el número de días `days` a la fecha inicial `startdate`
from_utc_timestamp(timestamp, string timezone)	`timestamp`	Asume que el `timestamp` indicado está según huso horario UTC (GMT) y lo convierte a la zona horaria indicada por `timezone`
to_utc_timestamp(string date, string timezone)	`timestamp`	Convierte la fecha dada por `date` en formato YYYY-MM-dd HH:mm:ss y supuesta en huso horario dado por `timezone` a su correspondiente *timestamp* en UTC
Funciones condicionales		
coalesce(T v1, T v2,..)	`T`	Devuelve el primer valor v que no es nulo, o nulo si todos los son. Es decir, si `v1` no es nulo, devuelve `v1`. Si `v1` es nulo y `v2` no es nulo, devuelve `v2`, y así sucesivamente
CASE A WHEN B THEN C ELSE D END	`T`	Si A = B, devuelve C, en caso contrario, devuelve D
CASE WHEN A THEN B ELSE C END	`T`	Si A = TRUE, devuelve B, en caso contrario, devuelve C
Funciones con *strings*		
concat(string A, string B,…)	`string`	Devuelve un único *string* formado por la concatenación de los *string* pasados como parámetros, en el orden en que fueron definidos
concat_ws(string SEP, array<string>)	`string`	Conforma un *string* mediante la concatenación de los *string* que conforman un *array*, separados por SEP
length(string A)	`int`	Devuelve el número de caracteres que forman el *string*, es decir su longitud
parse_url(string urlString, string partToExtract [, string keyToExtract])	`string`	Devuelve una parte específica de la URL especificada
split(string A, string pat)	`array`	Divide el *string* A por el carácter `pat`
substr(string A, int start, int len)	`string`	Devuelve los caracteres del *string* A comprendidos desde la posición `start` y `start + len`
UDAF		
count(*)	`bigint`	Devuelve el número de en-

		tradas que devolvería la consulta, incluso aquellas que contienen valores nulos
count(expr)	bigint	Devuelve el número de entradas para las cuales la expresión no devuelve valor nulo
sum(col)	double	Devuelve la suma de los elementos que forman el grupo col
avg(col)	double	Devuelve el valor medio de los elementos que forman el grupo col
min(col)	double	Devuelve el valor mínimo de los elementos que forman el grupo col
max(col)	double	Devuelve el valor máximo de los elementos que forman el grupo col
corr(col1, col2)	double	Calcula la correlación de *Pearson* del par de columnas col1 y col2 definidas para el grupo
collect_list(col)	array	Convierte una columna en un *array*
UDTF		
explode(array)	N registros	Devuelve un registro por cada uno de los elementos del *array*

Tabla 15. Funciones de *HiveQL*

Se puede encontrar más información en [199].

4.3.4. *UDFs*

Otro punto importante dentro de *Hive* es el uso de funciones definidas por el usuario *UDF*. Para poder usarlas, deberemos pasar por tres etapas. En primer lugar crearemos la función *UDF* que necesitamos porque *Hive* no nos proporciona lo que requerimos. Lo más común es implementar nuestra *UDF* en *Java*, siendo esta la opción que vamos a utilizar en este trabajo. Así, deberemos crear una clase *Java* en la que se implemente nuestra función. Supongamos que nuestra función se llama MyFunction, para implementarla seguiremos los siguientes pasos:

- Creamos una clase *Java*, denominada MyFunction, que extiende la clase *UDF*.
- Seleccionamos para ella un paquete adecuado, por ejemplo, com.hortonworks.hive.udf.
- Importamos las librerías necesarias para que sea reconocida como *UDF*.

- Dentro de ella creamos un método denominado `evaluate()`. Este método recibirá los parámetros de entrada correspondientes y devolverá el resultado que queremos obtener en *Hive*.

```
package com.hortonworks.hive.udf;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.hive.ql.exec.UDFArgumentException;

public class MyFunction extends UDF {

        public Tipo evaluate(Tipo1 variable1, Tipo2 variable2, …)
        throws Exception {
                …
                return variable;
        }
}
```

Una vez creada nuestra *UDF* la debemos registrar en *Hadoop*. Compilamos nuestra clase *Java* para obtener un ejecutable *JAR*. Si queremos que sea usada a través de un *cluster* de máquinas la opción ideal es alojarla en *HDFS*, ya que en caso de copiarla en el sistema de ficheros de la máquina, debería copiarse en todas las máquinas y exactamente en el mismo directorio. Para poder hacer uso de la nueva función, registramos el *JAR* en *Hive*:

```
ADD JAR hdfs://sandbox.hortonworks.com:8020/ruta/hacia/jar
```

Tras registrar el *JAR* creamos la función asociada (podemos crear un ejecutable con todas nuestras *UDF*). Podemos crearla definitivamente o de forma temporal. En nuestro caso usaremos funciones temporales, para que se borren cada vez que cerramos la sesión *Hive* y al iniciar de nuevo la sesión tengamos seguro cuál es la *UDF* que estamos utilizando (si se nos ha olvidado registrar nuestra función dará error, y así estamos seguros de que no estamos usando cualquier versión anterior de la misma función).

```
CREATE TEMPORARY FUNCTION myfunction AS 'com.hortonworks.hive.udf.MyFunction'
```

Tras esto, ya podemos usar nuestra *UDF*. Con respecto a estas sentencias, *HiveServer2* no soporta el comando ADD JAR. Al usar *HiveServer2*, es decir, *Hive* a través de los editores de *Apache Ambari* o *Hue*, no podemos usar esta sentencia. La alternativa que presentan es un formulario donde añadir el ejecutable *JAR* y definir la función *UDF*. En la versión de *HDP* que estamos utilizando (2.3.0) esta funcionalidad no funciona. En las versiones sucesivas sí que funciona correctamente. Debido a que se notó el error cuando la infraestructura ya estaba montada junto con el hecho de que para automatizar las consultas se deben usar las sentencias anteriores, las cuales sí funcionan si se lanzan desde línea de comandos (necesario para automatizar), no hicimos mayor caso a este error y, simplemente, usamos *Hive* por editor de línea de comandos cuando necesitamos usar una de nuestra *UDFs*.

Este es un tipo "simple" de funciones definidas por el usuario las cuales reciben un parámetro (uno o varios campos de una tabla, pero para cada columna solo una entrada) y devuelven un único elemento. Pero existen otras como *UDAF* y *UDTF.* Cualquiera de estos dos tipos de funciones es bastante más complejo de generar. En caso de necesitarse recurrir a la documentación correspondiente, ya que en este trabajo no se han utilizado, dada su complejidad.

Para la realización de nuestro trabajo, se han definidos tres *UDF*:

- `DateRange`. A partir de dos fechas que representan un inicio y fin de rango temporal, devuelve un *ArrayList* que contiene todos los días comprendidos entre ellas, ambas inclusive.
    - Parámetros de entrada:
        - Inicio: fecha de inicio, en formato YYYY-MM-dd.
        - Fin: fecha de fin, en formato YYYY-MM-dd.
    - Parámetro de salida: *ArrayList* que contiene todos los días comprendidos entre inicio y fin. Si la fecha de fin es anterior a la fecha de inicio, se devuelve un *ArrayList* vacío.
    - Ejemplo. Si inicio = '2016-01-01' y fin = '2016-01-03', el *ArrayList* contendrá los valores {'2016-01-01' , '2016-01-02', '2016-01-03'}
- `DayOfWeek`. A partir de una fecha, devuelve a qué día de la semana se corresponde.
    - Parámetro de entrada: fecha en formato YYYY-MM-dd.
    - Parámetro de salida: *String* que indica el día de la semana que se corresponde a la fecha obtenida como parámetro de entrada.
    - Ejemplo. Si la fecha de entrada es '2016-02-26' la salida será 'VIERNES'
- `ExtractCourseID`. A partir de un identificador de curso *META*, según la estructura del *Moodle* de la UVa, obtiene su parte histórica, es decir, sin la referencia al año académico.
    - Parámetro de entrada: identificador META del curso
    - Parámetro de salida: identificador histórico del curso
    - Ejemplo. Dado `curso1-2014//curso2-2014` como identificador de curso META, devolverá `curso1//curso2`.

Esta última *UDF* se entenderá cuando se explique el manejo de identificadores de curso en el análisis *BigData.* Estas *UDF* están compiladas en el fichero `HiveUDF.jar` en el directorio *lib* del código fuente proporcionado junto con este documento.

4.3.5. Estructuras para manipular datos

HiveQL proporciona distintas estructuras para manipular los datos. Explicar todas en este texto podría ser extenso, por lo que vamos a explicar tres de ellas, que han resultado de vital importancia en el desarrollo del trabajo. Se van a explicar, en el contexto de cómo se han utilizado o cómo se ha interpretado que es su comportamiento para lograr nuestros objetivos.

Por un lado, *HiveQL* proporciona distintas declaraciones para la unión (*JOIN*) de tablas. A esta operación, en nuestro trabajo, la hemos referenciado continuamente con el nombre de "cruce de tablas" debido a que lo que realmente implementa es construir una tabla final a partir de columnas de dos o más tablas originales. La operación de unión de tablas en *HiveQL*,

que se denomina *UNION*, se encarga de juntar los registros de dos resultados o tablas que tengan exactamente la misma estructura, como si fuera añadir a uno los registros del otro.

Si nos centramos en lo que hemos denominado "cruce de tablas" hemos utilizado dos sentencias con dos objetivos distintos. Las dos se basan en lo mismo pero con una ligera diferencia. Supongamos que tenemos dos tablas A y B con las estructuras dadas en la Tabla 16, es decir, una almacena información de los accesos totales por curso – usuario, y otra solo los accesos a módulos.

TABLA A	TABLA B
curso	curso
usuario	usuario
num_accesos_totales	num_accesos_modulos

Tabla 16. Estructuras de las tablas de ejemplo para explicar *LEFT OUTER JOIN* y *CROSS JOIN*

Puede que necesitemos generar una única tabla en la que guardar la información de ambas tablas, es decir, una tabla, en la que para cada curso – usuario guardemos el número de accesos totales y el número de accesos a módulos. Partiendo de A y B podemos hacerlo por medio de un cruce de tablas. En todo cruce de tablas el punto clave está en los campos que se usan para el cruce, los cuales indican qué criterios se van a usar para cruzar las tablas. Definamos los dos tipos de cruces que vamos a utilizar:

- `LEFT OUTER JOIN`: esta sentencia toma como referencia una tabla o resultado, la parte izquierda (*LEFT*) de la consulta, y trata de añadir a su derecha columnas tomadas de otras tablas. De esta forma, los registros de la tabla de salida vienen marcados por los registros de la tabla izquierda. La recíproca se denomina `RIGHT OUTER JOIN`.

- `CROSS JOIN`: es parecida a la anterior, pero con la diferencia en que no toma ninguna tabla como referencia, así, los registros de salida serán aquellos que están presentes en ambas tablas.

Se entiende mejor con un ejemplo, supongamos que A presenta el contenido de la Tabla 17 y B de la Tabla 18.

curso	usuario	num_accesos_totales
Curso1	**Usuario1**	2
Curso1	**Usuario2**	4
Curso2	**Usuario1**	80
Curso2	**Usuario2**	3
Curso3	**Usuario1**	0

Tabla 17. Ejemplo de contenido para la Tabla A para la explicación de `LEFT OUTER JOIN` y `CROSS JOIN`

curso	usuario	num_accesos_modulos
Curso2	**Usuario1**	20
Curso2	**Usuario3**	2

Tabla 18. Ejemplo de contenido para la Tabla B para la explicación de `LEFT OUTER JOIN` y `CROSS JOIN`

En este ejemplo, queremos resultados para cada clave `curso – usuario`, por lo que el cruce de tablas se realizará por medio de estos dos campos. Empecemos por un cruce por medio de `LEFT OUTER JOIN` (representado en la Figura 86), mediante el siguiente código:

```
SELECT A.curso,
    A.usuario,
    A.num_accesos_totales,
    B.num_accesos_modulos
FROM (
    SELECT *
    FROM tablaA
    ) A
LEFT OUTER JOIN (
    SELECT *
    FROM tablaB
    ) B ON (A.curso = B.curso)
    AND (A.usuario = B.usuario)
```

Dada la definición, la tabla que se está tomando como parte izquierda es la primera definida, es decir, A. Esto implica, que todas las combinaciones `curso – usuario` que se tendrán en cuenta para conformar el resultado final serán las que aparezcan en A. A mayores de la columna que trae A, se trata de añadir la que trae B, pero si en B no se encuentran datos, se dejará vacío. Para evitarlo se puede usar la función COALESCE, de la forma `COALESCE(B.num_accesos_modulos, 0)` para que si no encuentra entrada para una combinación `curso – usuario` en B, la complete con 0. El resultado obtenido se muestra en la Tabla 19. Se puede apreciar como solo aparecen las combinaciones `curso – usuario` de A, ignorándose todas aquellas de B que no están en A.

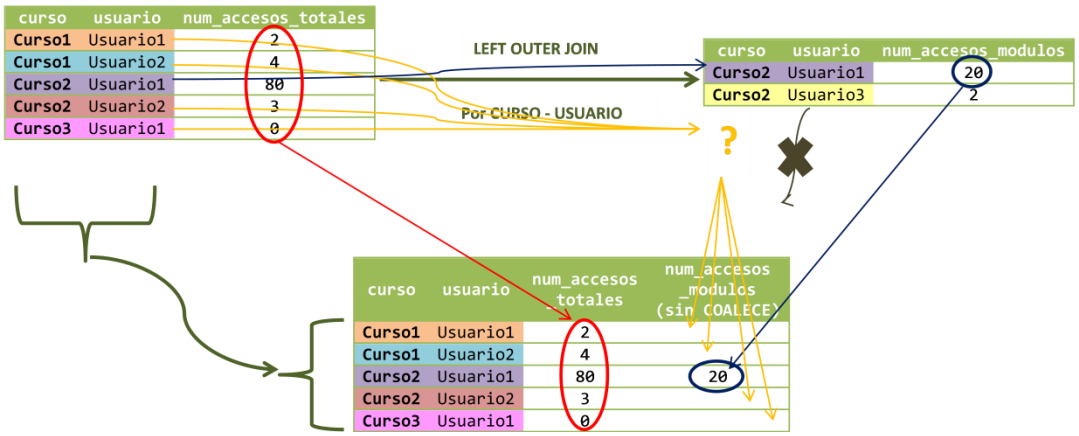


Figura 86. Representación gráfica del ejemplo para la explicación de `LEFT OUTER JOIN` I

curso	usuario	num_accesos_totales	num_accesos_modulos (sin COALECE)	num_accesos_modulos (con COALECE)
Curso1	**Usuario1**	2		0
Curso1	**Usuario2**	4		0
Curso2	**Usuario1**	80	20	20
Curso2	**Usuario2**	3		0
Curso3	**Usuario1**	0		0

Tabla 19. Resultado `LEFT OUTER JOIN` sobre ejemplo I

Si usaramos B como parte izquierda de la consulta, obtendríamos el resultado de la Tabla 20.

curso	usuario	num_accesos_totales (sin COALESCE)	num_accesos_totales (con COALESCE)	num_accesos_modulos
Curso2	Usuario1	80	80	20
Curso2	Usuario3		0	2

Tabla 20. Resultado LEFT OUTER JOIN sobre ejemplo II

Si para cruzar las tablas usaramos un CROSS JOIN (Figura 87):

```
SELECT A.curso,
    A.usuario,
    A.num_accesos_totales,
    B.num_accesos_modulos
FROM (
    SELECT *
    FROM tablaA
    ) A
CROSS JOIN (
    SELECT *
    FROM tablaB
    ) B ON (A.curso = B.curso)
    AND (A.usuario = B.usuario)
```

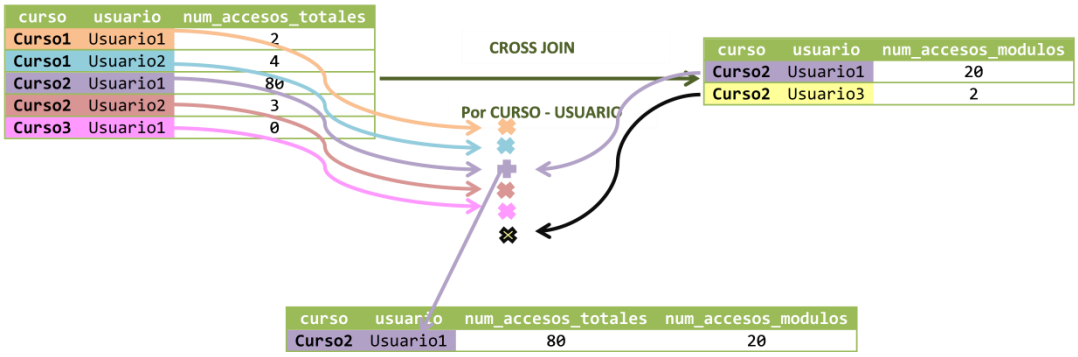


Figura 87. Representación gráfica del ejemplo para la explicación de CROSS JOIN I

Obtendríamos como resultado solo las combinaciones `curso – usuario` que estuvieran en A y B, es decir, el resultado de la Tabla 21. En este caso, da igual el orden del cruce, el resultado sería el mismo, ya que se queda solo con lo común.

curso	usuario	num_accesos_totales	num_accesos_modulos
Curso2	Usuario1	80	20

Tabla 21. Resultado CROSS JOIN sobre ejemplo I y II

En este ejemplo, no haría falta, porque si hemos capturado la métrica para una combinación `curso – usuario`, esta tendrá un valor, y no aparecerán valores vacíos, pero se puede usar la función COALESCE(), exactamente igual que en LEFT OUTER JOIN.

El cruce de tablas también se puede realizar con tablas que tengan estructuras un poco más distintas, por ejemplo cruzar la tabla A anterior, con la tabla C que guarda para cada curso, el total de accesos de todos sus usuarios, como se muestra en la Tabla 22.

curso	num_accesos_todos_usuarios
Curso1	10
Curso2	100
Curso4	20

Tabla 22. Ejemplo de contenido para la Tabla C para la explicación de LEFT OUTER JOIN y CROSS JOIN

Puede que necesitemos cruzar ambos resultados para calcular, por ejemplo, el porcentaje de accesos que corresponde a cada usuario. Así a cada combinación `curso – usuario` le debemos añadir el total de accesos de todos los usuarios al curso. Podemos cruzar ambas tablas por medio de un `LEFT OUTER JOIN` y tomando como clave, solo el campo `curso`, ya que es lo único que comparten (Figura 88):

```
SELECT A.curso,
    A.usuario,
    A.num_accesos_totales,
    COALESCE(C.num_accesos_todos_usuarios,0)
FROM (
    SELECT *
    FROM tablaA
    ) A
LEFT OUTER JOIN (
    SELECT *
    FROM tablaC
    ) C ON (A.curso = C.curso)
```

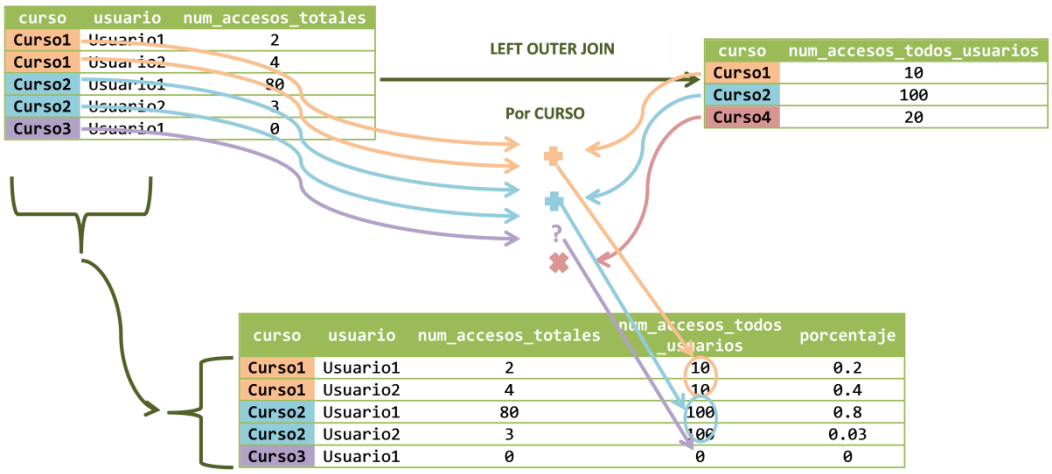


Figura 88. Representación gráfica del ejemplo para la explicación de LEFT OUTER JOIN II

Obteniendo el resultado de la Tabla 23.

curso	usuario	num_accesos_totales	num_accesos_todos_usuarios	porcentaje
Curso1	Usuario1	2	10	0.2
Curso1	Usuario2	4	10	0.4
Curso2	Usuario1	80	100	0.8
Curso2	Usuario2	3	100	0.03
Curso3	Usuario1	0	0	0

Tabla 23. Resultado LEFT OUTER JOIN sobre ejemplo III

Si los cruzáramos por medio de un `CROSS JOIN` (Figura 89), perderíamos el resultado sobre el `Curso3`, ya que no tenemos el número total de accesos de todos los usuarios en la Tabla C (Tabla 24).

curso	usuario	num_accesos_totales	num_accesos_todos_usuarios	porcentaje
Curso1	Usuario1	2	10	0.2
Curso1	Usuario2	4	10	0.4
Curso2	Usuario1	80	100	0.8
Curso2	Usuario2	3	100	0.03

Tabla 24. Resultado CROSS JOIN sobre ejemplo III

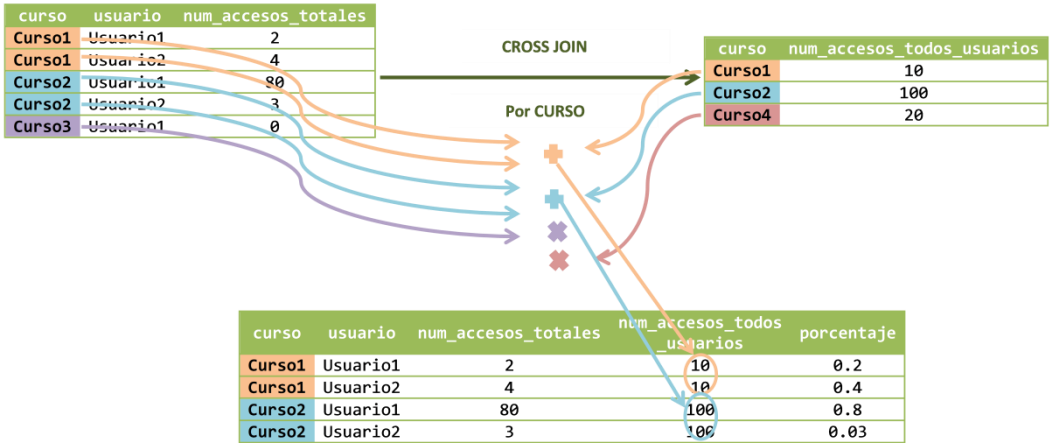


Figura 89. Representación gráfica del ejemplo para la explicación de CROSS JOIN II

Por otro lado, destacamos la sentencia `LATERAL VIEW` [200]. Esta se usa en conjunto con otras funciones *UDTF*, sobre todo con la función `EXPLODE()`, como va ser nuestro caso. Una función *UDTF* generará una o más filas para cada elemento de entrada. `LATERAL VIEW` primero se encargará de aplicar la *UDTF* correspondiente, en nuestro caso `EXPLODE()` y después se encargará de unir cada una de las filas resultantes con el valor original que le correspondía. Se entiende mejor con un ejemplo. Pongámonos bajo uno de los supuestos en que vamos a utilizar la función: dada una combinación `curso – usuario`, tenemos para ella dos fechas, una de inicio y otra fin (por ejemplo, como en la Tabla 25), y queremos generar todas las combinaciones `día – curso – usuario` para cada `curso – usuario` con todos los días comprendidos entre ambas fechas. Tenemos nuestra *UDF* `DateRange` que dadas dos fechas, nos devuelve un *array* con todas las fechas comprendidas entre ellas, ambas inclusive.

curso	usuario	inicio	fin
Curso1	Usuario1	2016-01-01	2016-01-05
Curso1	Usuario2	2016-01-03	2016-01-07

Tabla 25. Tabla de ejemplo para la explicación de LATERAL VIEW

Aplicando el siguiente código:

```
SELECT dia,
    A.curso,
    A.usuario
FROM tablaA A LATERAL VIEW explode(daterange(A.inicio, A.final))
subview AS dia;
```

Obtendremos el resultado de la Tabla 26.

dia	curso	usuario
2016-01-01	Curso1	Usuario1
2016-01-02	Curso1	Usuario1
…		
2016-01-05	Curso1	Usuario1
2016-01-03	Curso1	Usuario2
2016-01-04	Curso1	Usuario2
…		
2016-01-07	Curso1	Usuario2

Tabla 26. Resultado LATERAL VIEW sobre ejemplo

Es decir, para cada combinación `curso – usuario`, primero se aplica la función `DateRange` para sus valores de inicio y fin. Después, se pasa este resultado por la función `EXPLODE()` la cual generará un columna con el *array* resultante de `DateRange`. Por último a cada una de las entradas de esta columna les añade las columnas `curso` y `usuario`, cuyo valor proviene de la combinación `curso – usuario` que le corresponde al origen de la columna. Se puede entender en el gráfico de la Figura 90.

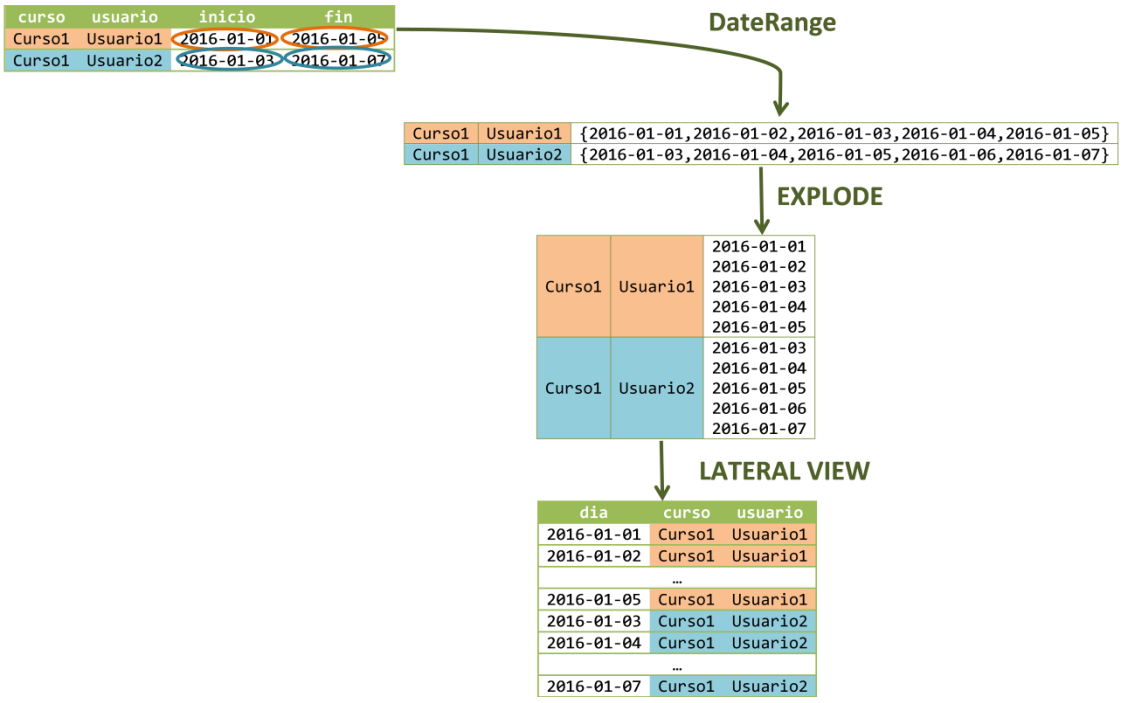


Figura 90. Representación gráfica del ejemplo para la explicación de LATERAL VIEW

4.3.6. *Bucketing & Partitioning*

Para gestionar tablas, *Hive* proporciona muchas alternativas, pero en nuestro caso nos hemos centrado en dos, que son las más usuales: el *bucketing* y el particionado (*partitioning*) de tablas.

El **particionado** o *partitioning* [201], [202] de tablas consiste en dividir las tablas en varias partes, en base al valor de una o varias columnas, para generar una estructura de almacenamiento más jerárquica. *Hive* almacena los datos de las tablas en ficheros en *HDFS*. A cada tabla le corresponde un determinado directorio. El particionado va a generar una nueva estructura de directorios dentro del directorio de la tabla, de tal forma que, registros que tengan distinto valor para una (o más de una) determinada columna, se guarden por separado. Por ejemplo, supongamos que tenemos una tabla que almacena información de accesos por curso – usuario y particionamos la tabla por curso, esto implica, que la información de los usuarios de distintos cursos se guarda por separado (Figura 91), lo cual va a agilizar las búsquedas de información dentro de cada curso, ya que cuando queramos obtener información sobre un determinado curso, solo miraremos su directorio asociado, y no el de los demás. Sin particionar las tablas, se deberían mirar todos los ficheros para analizar un solo curso, ya que la información de los distintos cursos estaría mezclada entre los distintos ficheros (Figura 92).

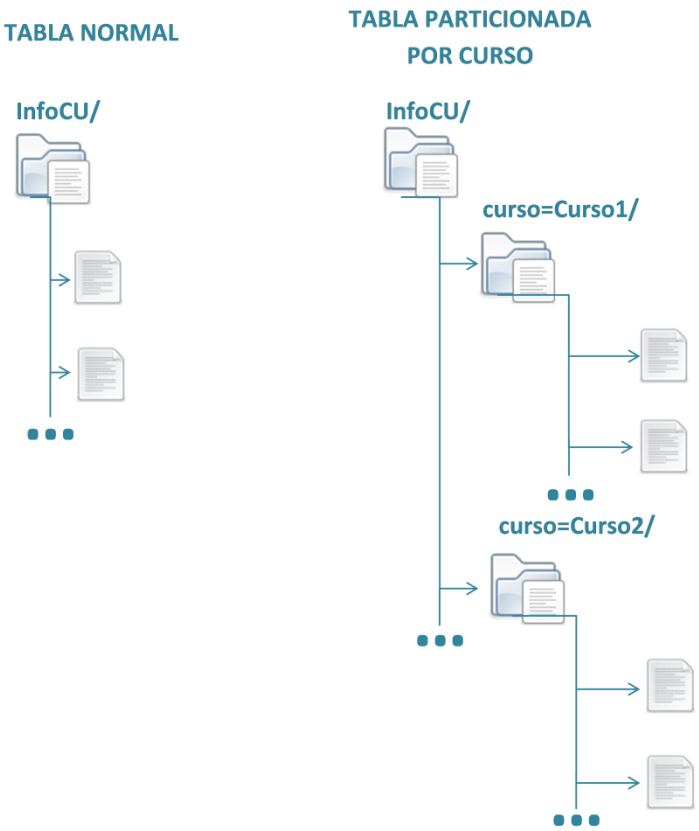


Figura 91. Ejemplo de estructuración de datos con particionado de tablas en *Hive* I

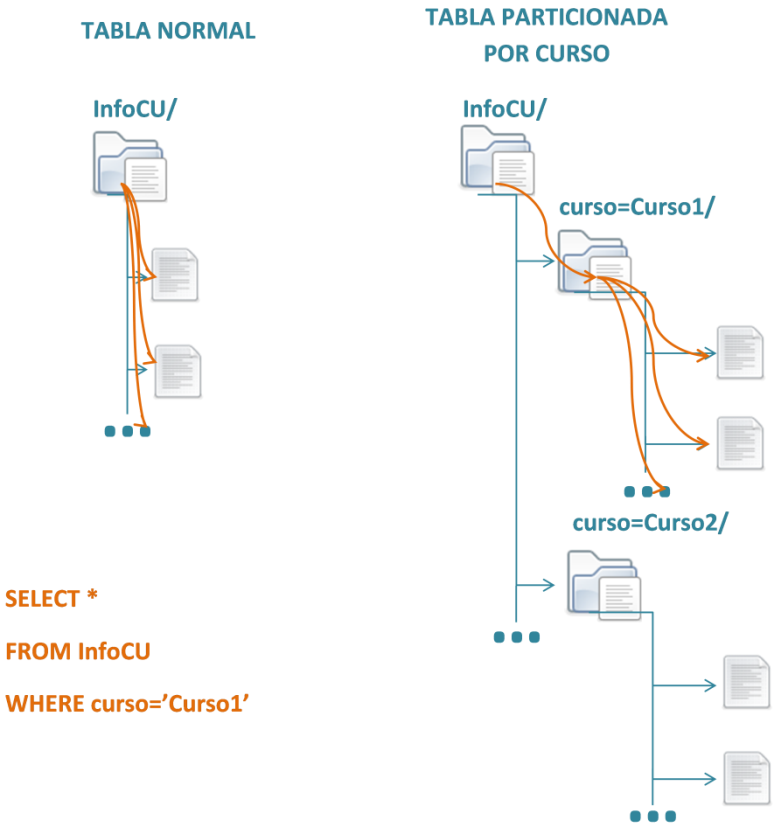


Figura 92. Ejemplo de recorrido de directorios para la ejecución de sentencias *Hive* sobre una tabla particionada

Las tablas se pueden particionar por múltiples campos, por ejemplo, el particionado se utiliza sobre todo (aunque no va a ser nuestro caso) para guardar información referenciada por fecha de tal forma que se genere una estructura jerárquica con un primer nivel que identifica el año, dentro de cada año, un segundo nivel que identifica el mes, y dentro de cada mes, un tercer nivel que identifica el día, como se muestra en la Figura 93. Así cuando queramos revisar los datos de un día concreto, se recorrería el árbol de directorio hasta llegar al directorio que corresponde.

Ventajas del particionado:

- Permite distribuir la carga horizontalmente.
- Permite optimizar las sentencias con cláusulas `WHERE` ya que ya no se obliga a recorrer todos los ficheros.

Limitaciones del particionado:

- Si una tabla genera muchas particiones, esto creará gran cantidad de ficheros y directorios en *HDFS*, sobrecargando al *NameNode* ya que deberá guardar los metadatos correspondientes.
- Se optimiza el uso de cláusulas `WHERE`, pero puede provocar pérdida de rendimiento en la ejecución de cláusulas `GROUP BY`.

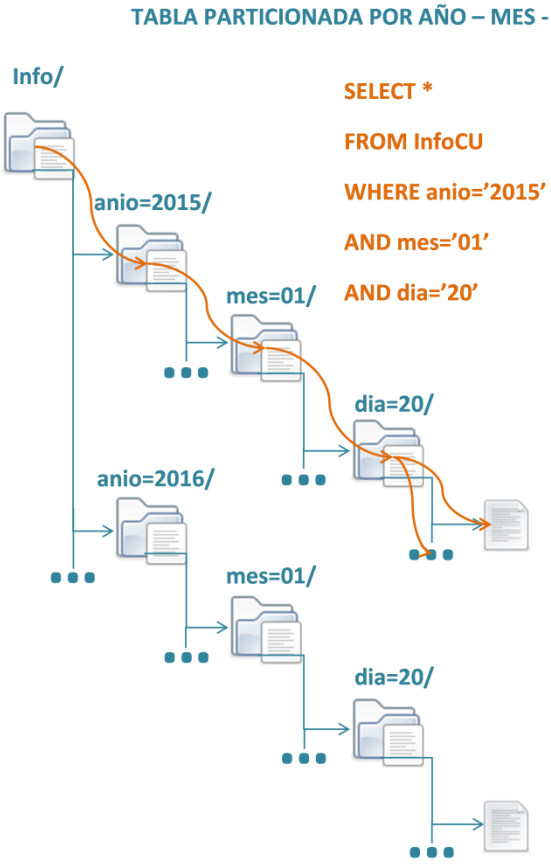


Figura 93. Ejemplo de estructuración de datos con particionado de tablas en *Hive* II

Además, el particionado lleva dos **modos de funcionamiento**: **estricto** (*strict*) y **no estricto** (*nonstrict*), uno de los cuáles debe ser fijado en la configuración de *Hive*. El modo estricto no permite realizar consultas a tablas particionadas si no se selecciona una o varias particiones, es decir, no se incluye una cláusula `WHERE`. El modo no estricto, permite realizar cualquier consulta. Por ello, se deberá tener en cuenta qué necesitamos de las tablas particionadas para fijar el modo correcto.

La creación de una tabla particionada es igual al de una tabla normal, pero añadiendo la opción de particionado e indicando la/las columna/s de particionado, por ejemplo, mediante un código de la siguiente forma:

```
CREATE TABLE moodle2_bigdata_results.matriculaciones (
    curso string,
    usuario INT,
    inicio DATE,
    fin DATE
    ) PARTITIONED BY (
    STATUS string,
    activo string
    ) STORED AS ORC;
```

Los campos de particionado no se indican en la definición de columnas, sino en la definición del particionado, y su orden de definición es importante, por ejemplo, en este caso, el primer orden de jerarquía sería por la clave `status` y dentro de ella, por `activo`.

A la hora de insertar datos en una tabla particionada tenemos dos opciones dados dos tipos de particionado. El primero es el **particionado estático**, el cual no nos es de utilidad en este trabajo, ya que en cada sentencia `INSERT` debemos indicar la partición en la que queremos insertar los datos, de la forma:

```
INSERT INTO moodle2_bigdata_results.matriculaciones
   PARTITION (status='PROFESOR', activo='ACTIVO')
   VALUES ('Curso1','Usuario1','2016-01-01','2016-01-05')
```

Es decir, debemos dar nombre a la partición, pero eso no nos es posible en nuestro caso, ya que este parámetro dependerá de cada caso concreto en que queramos insertar algo, que después estará automatizado.

Por ello, vamos a usar el **particionado dinámico**, en el que tan solo especificados la clave de partición, y después le pasamos todos los campos del registro que queremos insertar, pasando los valores para la clave de la partición en último lugar, es decir, de la forma:

```
INSERT INTO moodle2_bigdata_results.matriculaciones
   PARTITION (status, activo)
   VALUES ('Curso1', 'Usuario1', '2016-01-01', '2016-01-05', 'PROFESOR', 'ACTIVO')
```

Para poder usar el particionado dinámico, se deben fijar las siguientes propiedades en *Hive*:

- `hive.exec.dynamic.partition=true`.
- `hive.exec.dynamic.partition.mode=nonstrict`.

En cualquiera de los casos, tener en cuenta que la columna particionada no se almacena con el resto de la información dentro del fichero o ficheros que correspondan, ya que será el nombre del directorio que contiene los ficheros el que marque el valor de ese campo, por ejemplo, el registro anterior generaría la estructura de la Figura 94.

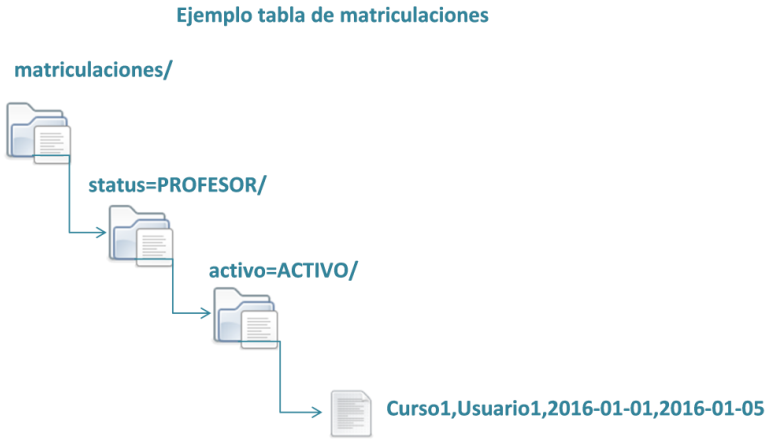


Figura 94. Ejemplo de estructuración de datos con particionado de tablas en *Hive* III

Para manejar las particiones se debe tener en cuenta:

- No podemos modificar los campos particionados, es decir, si queremos realizar una operación UPDATE sobre un registro de la tabla, esta no puede cambiar el valor de ningún campo de particionado.

- Se pueden borrar particiones mediante una sentencia de la forma:

```
ALTER TABLE moodle2_bigdata_results.matriculaciones
   DROP PARTITION (curso='Curso1')
```

Por otro lado, tenemos el ***Bucketing*** [202], [203] de tablas. Este se presenta como una alternativa a ciertos casos en los que el particionado no es bueno, principalmente debido a que:

- El número máximo de particiones por tabla está limitado.

- No se asegura que todas las particiones tengan una cantidad (tamaño) de datos equivalente, ya que depende del valor de los datos.

Para poder definir un *bucket* sobre una tabla, también necesitamos escoger sobre qué columnas se va a hacer el *bucket*, y el número total de *buckets* que queremos tener. Cuando se añade un nuevo dato a la tabla, se le debe asignar un *bucket*, básicamente un número de *bucket* de entre los que hemos definidos. Para ello, aplica una función *Hash* a las claves del *bucket* y calcula:

(función hash sobre las columnas del bucket) mod (número de buckets)

Para así obtener el número de *bucket* que le corresponde. Así se garantiza, que todos los datos con la misma combinación de columnas del *bucket* van a parar al mismo *bucket*, y si se llega a configurar de forma correcta el número de *buckets* con respecto a las diferentes combinaciones de las columnas, puede que cada *bucket* aloje una sola combinación.

Ventajas del *bucketing*:

- Se consigue optimizar las consultas, sobre todo las que contienen sentencias `JOIN` o `GROUP BY` por las claves del *bucket*. Es decir, se puede optimizar una sentencia `JOIN` si las N tablas que vamos a cruzar tienen realizado un *bucket* por el conjunto de claves que se usan para el cruce, así como que existe proporcionalidad entre el número de *buckets* definidos. Por otro lado, una sentencia de `GROUP BY` se puede agilizar si la tabla tiene hecho un *bucket* por el conjunto de claves por las que se va a agrupar.

- Permite optimizar no solo el tiempo de ejecución si no también la memoria necesitada.

Limitaciones del *bucketing*:

- Empeora el rendimiento de las sentencias con cláusulas `WHERE`, sobre tener la tabla sin *bucketing* y sin *particionar*.

Una tabla con *bucket* se puede crear de la siguiente forma:

```
CREATE TABLE moodle2_bigdata_results.matriculaciones (
    curso string,
    usuario INT,
    inicio DATE,
    fin DATE
```

```
    )
CLUSTERED BY (
    curso,
    usuario
    )
   INTO 200 BUCKETS STORED AS ORC TBLPROPERTIES('transactional' =
'true');
```

En este caso, las columnas del *bucket* se deben definir con el resto de columnas. No se ha encontrado información al respecto de un distinto comportamiento o rendimiento por el orden de definición de las claves del *bucket*. En este caso, 200 sería el total de *buckets* a definir. Además, para esta tabla se activa el comportamiento transaccional (tratado más adelante).

La inserción de datos a una tabla con *bucket* es igual a la inserción normal, no hay que especificar nada especial como en el caso de la partición. Además, en este caso, la información de las columnas del *bucket* sí que se guarda con el resto de la información, ya que no se puede asegurar que cada *bucket* sea un valor concreto para ellas y el *bucket* no guarda información de las columnas almacenadas o posibles de almacenar, ya que simplemente guarda un número identificador de *bucket*

A mayores, se pueden **mezclar ambas técnicas**, es decir, particionar una tabla por unos campos y hacerle el *bucket* por otros, siempre que no sean los mismos, por ejemplo:

```
CREATE TABLE moodle2_bigdata_results.matriculaciones (
    curso string,
    usuario INT,
    inicio DATE,
    fin DATE
    ) PARTITIONED BY (
    STATUS string,
    activo string
    ) CLUSTERED BY (
    curso,
    usuario
    )
   INTO 200 BUCKETS STORED AS ORC TBLPROPERTIES('transactional' =
'true');
```

En este caso, hemos particionado la tabla, primero por el campo `status` y después por `activo`, y dentro de cada partición haremos un *bucket* por los campos `curso` y `usuario`.

Centrándonos en **nuestro trabajo**:

- Haremos *bucketing* en aquellas tablas que después vayan a pasar por una etapa de procesamiento con `JOINs` o `GROUP BY`, atendiendo a las claves de esta operación. En muchas ocasiones nos hemos encontrado con que una misma tabla requeriría de *bucketing* por distintas claves. En estos casos, se ha tratado de optimizar la consulta que se cree sería más conflictiva en cuanto a carga. En otros casos en los que no ha sido posible tomar esta decisión, se ha dejado la tabla sin *bucket*.

- Haremos un particionado de aquellas tablas sobre las que es sistemática la realización de sentencias con cláusulas `WHERE`, por los campos de filtrado, siempre que estos no fueran campos necesarios para el *bucket*.

- También se ha realizado un particionado de las tablas que almacenan la información de los resultados finales de nuestro análisis, para llevar un registro de todos los análisis realizados, es decir, cada vez que se genere un análisis, que para cada tabla genera una serie de valores obtenidos de *Moodle*, guardaremos esa información en la partición correspondiente al día del análisis, para poder ver la evolución de estos resultados. Se realiza solo en los casos finales e interesantes de ver la evolución, no en todas las tablas generadas.

Durante la realización de pruebas se ha comprobado cierto comportamiento asociado a estas técnicas:

- Cuando se guarda información en formato *bucket*, el procesado *Hive* se ve incrementado ya que debe generar los N *buckets* que le hayamos indicado. A pesar de que con la información que tengamos no se puedan completar los N *buckets*, genera los N ficheros correspondientes, aunque solo algunos contendrán información. Por ello, en diversos tutoriales, se recomienda no hacer *bucket* en exceso, solo en aquellos casos en los que queramos optimizar una sentencia `JOIN` o `GROUP BY`.

- Cuando se realizan varios `INSERT` sobre una tabla con *bucket*, por cada `INSERT` se genera un *bucket* con la información de los X registros que tuviera esa sentencia. Si tenemos activado el comportamiento transaccional (como en el ejemplo anterior) cada *bucket* se almacena en un directorio *Delta*, *Hive* presenta un *Compactor* que se encarga, cada cierto tiempo, de juntar la información de todos estos *Deltas*. Si no se tienen activadas las transacciones para la tabla, la información se almacena directamente en el directorio de la tabla, pero múltiples `INSERT` seguirán generando nuevos ficheros, por ejemplo, un primer `INSERT` genera el fichero correspondiente al primer *bucket* bajo el nombre `bucket00000`, un segundo `INSERT` generará un segundo fichero, para el mismo *bucket*, bajo el nombre `bucket00000_copy`, pero como en este caso se tienen deshabilitadas las transacciones, es de suponer que el *Compactor* no entre en juego y los ficheros nunca se unan.

  En un principio se pensó en deshabilitar las transacciones ya que generar esos múltiples directorios obliga a *Hive* a tener que recorrerlos cuando se lance cualquier sentencia, lo cual en principio parece más costoso que solo recorrer un directorio para encontrar qué contiene cada *bucket*. Sorprendió el comportamiento no transaccional (se repite en todas las tablas, cuando se añade nueva información, se añade a nuevos ficheros), ya que al final por cada `INSERT` tendremos un fichero para cada *bucket*, aunque en este caso, en el mismo directorio. Pero a mayores se encontró otro problema por el que se volvió a las tablas con comportamiento transaccional: aumentaban los requisitos de memoria necesaria para realizar procesamientos que activando las transacciones se realizaban "de sobra".

- Guardar información "muy particionada", es decir, con muchas particiones, requiere de mucho tiempo, y puede provocar errores de falta de memoria en *Hive*, por ello, se ha marcado como pauta usar particionado en casos en los que no se generen muchas particiones al guardar la información. Puede que se generen muchas particiones en total, pero que cada sentencia de inserción INSERT trate con unas pocas de ellas.

- No se ha sabido escoger cuál es el número de *buckets* idóneo para definir las tablas. Se ha escogido 200 en todos los casos, por marcar un valor, y probar las sentencias. Convendría realizar una análisis real de la cantidad de datos (principalmente cuántas claves distintas de *bucket* se van a tener) para ajustar estos valores. La complejidad ha venido dada en que para que realmente el *bucket* actúe de forma eficiente durante el cruzado

(JOIN) de tablas (operación más costosa en *Hive*), se necesita que las tablas que se crucen tengan un número proporcional de *buckets*; y la tarea de encontrar valores que se ajustasen a la previsión de datos que se van a tener en cada tabla y que luego coincidiesen entre ellos, es bastante laboriosa.

4.3.7. Conexión con bases de datos externas

Hive se puede conectar con bases de datos externas, en el sentido de que las tablas y su correspondiente información se mantienen en otro almacén, pero *Hive* accede a su información mediante las sentencias típicas, y en ciertos casos también puede añadir información por medio de sentencias INSERT. En estos casos se usan tablas externas, aunque no es su único uso.

En *Hive* se pueden definir tablas **internas** o **externas**. Las primeras, son las "normales" o típicas, aquellas cuya información y metadatos asociados se guardan en *Hive* (y *HDFS*). Las tablas externas también pueden ser tablas únicas de *Hive*, pero sobre las que *Hive* no debería tener control completo, principalmente, los ficheros que contienen los datos no se eliminan aunque *Hive* ejecute una sentencia DROP TABLE. Este tipo de tablas se usan cuando los datos de *Hive* son usados por otros procesos. El otro tipo de tablas externas son las que se refieren a datos que no están en *Hive*, por ejemplo, en nuestro trabajo usamos tablas externas para transferir la información a *ElasticSearch*, es decir, una especie de conexión entre *Hive* y unos datos que se encuentran en otra base de datos. No es correcto decir siempre tabla externa, ya que, por ejemplo, en *ElasticSearch* no hay tablas, sino índices. A través de estas tablas "conexión" *Hive* puede leer y escribir datos en bases de datos externas, pero nunca puede ejecutar una sentencia DROP TABLE sobre ellas, es decir, no puede borrar la tabla, ya que al ser una tabla externa, no tiene control de gestión sobre ellas. Dependiendo del tipo de base de datos externa, se podrán realizar distintas operaciones de manipulación de los datos más allá de leer e insertar, por ejemplo, en *ElasticSearch* no está permitido el uso de sentencias DELETE para borrar datos, principalmente, porque para borrar un documento del índice (lo que se correspondería a un registro de la tabla "conexión") hay que borrar el índice entero, es decir, no soporta borrar documentos de un índice.

En un principio, se dudó en usar una tabla externa sobre la base de datos de *Moodle* para la lectura de datos, para así evitar la transferencia mediante *Sqoop*, muy costosa en tiempo. Pero finalmente se declinó la alternativa, ya que con *Sqoop* podemos asegurar una copia de las tablas realizadas a una hora concreta y que usamos para realizar el análisis, es decir, si usamos una misma tabla en múltiples puntos aseguramos que es la misma. Mientras que usando una tabla externa, no podríamos asegurarlo ni cuantificar los errores que podrían aparecer si entre un uso y otro de la misma tabla, cambia su información.

En cualquier caso, la creación de una tabla externa, lleva asociado la sentencia CREATE EXTERNAL TABLE. La consulta necesaria dependerá de la tabla concreta que queramos crear, pero principalmente:

- Deberemos indicar cómo la queremos almacenar. Es decir, qué tipo de almacenamiento es, por ejemplo, a qué tipo de base de datos externa nos estamos refiriendo. Se especifica mediante STORED BY seguido de:
    - `'org.elasticsearch.hadoop.hive.EsStorageHandler'`, si es con *ElasticSearch*

  - `'org.apache.hadoop.hive.hbase.HBaseStorageHandler'`, si es con *HBase*
  - *ORC*, si es en el propio *Hive*.
  - Etc.

- Dependiendo del tipo de almacenamiento, deberemos añadir información adicional, con `TBLPROPERTIES`, y en algunos casos con `WITH SERDEPROPERTIES`. Por ejemplo, en *ElasticSearch*, debemos indicar el *host* que aloja la base de datos, y el índice al que va a conectar esta tabla.

En cada caso concreto se deberá recurrir a la documentación correspondiente para ver las variantes. Por ejemplo, en el caso de *ElasticSearch*, se especifica en el apartado 4.1.6.

4.4. Estructuración del código fuente

Antes de comenzar a analizar el trabajo realizado, se plantea la estructuración del código fuente proporcionado junto con este documento.

En la carpeta denominada *CódigoFuente* se puede encontrar el siguiente contenido:

- *apps*: aplicaciones para la ejecución de los programas que más adelante se detallan.
- *conf*: ejemplos de ficheros de configuración necesarios para la ejecución de los programas anteriores (ver secciones 4.1.2 y 4.8).
- *dataset*: ficheros donde se definen los *DataSet* utilizar en los programas anteriores (ver sección 4.8, en concreto el apartado 4.8.3).
- *Driver Sqoop*: Distintos *drivers* para la conexión de *Sqoop* con bases de datos *SQL* típicas.
- *ES – Kibana*: ficheros relacionados con la configuración de *ElasticSearch* y *Kibana* (ver sección 4.1.6).
- *Java. Código Fuente*: código fuente de los distintos programas y funciones *Java* implementados para la ejecución de las aplicaciones, junto con información sobre su compilación.
- *lib*: librerías necesarias para la ejecución de las aplicaciones.

Dentro de *apps* nos encontramos dos subdirectorios: *inicial* e *incremental* donde se desarrolla la ejecución inicial o incremental, respectivamente, para la consecución de objetivos (ver sección 4.6). Dentro de cada una de estas aplicaciones, nos encontramos 87 carpetas y un fichero `bundle.xml`. El fichero define el paquete de ejecución de la aplicación (ver sección 4.8, en concreto el apartado 4.8.4). Cada una de las restantes carpetas define un bloque de ejecución utilizado para la consecución de un resultado concreto (ver secciones 4.6 y 4.7). Dentro de cada una de ellas nos encontramos:

- Uno o varios (según corresponda) *scripts Hive* que contienen las consultas *HiveQL* necesarias para la obtención del resultado.

- Un *script* en *Shell* denominado `createSuccessFlag.sh` utilizado para la automatización de la ejecución condicional necesaria (ver sección 4.8, en concreto el apartado 4.8.3).

- Un fichero `workflow.xml` donde se define el flujo de trabajo necesario para la automatización de la ejecución de este bloque (ver sección 4.8, en concreto el apartado 4.8.2).

- Un fichero `coordinator.xml` donde se define el calendario de lanzamiento y la ejecución condicional del bloque (ver sección 4.8.3, en concreto el apartado 4.8.3).

- En ciertos bloques dedicados a la transferencia de datos a *ElasticSearch* (ver sección 4.7) se encontrará otro *script* en *Shell* dedicado a la creación del índice correspondiente en *ElasticSearch* mediante la herramienta `curl`.

- Otros casos:

    - El primer bloque de ejecución en la aplicación incremental, `DatosOrigen`, se encarga de borrar el *flag* de finalización del análisis anterior, mediante un segundo *script* en *Shell*. El último bloque, en cualquiera de las dos aplicaciones, `Finalizado`, se encarga de borrar todos los *flags* de completitud generados durante la ejecución de la aplicación, en el mismo *script* `createSuccessFlag.sh` (ver sección 4.8, en concreto el apartado 4.8.3).

Para la ejecución de estas aplicaciones se debe crear en *HDFS* la estructura de directorios planteada en la Figura 95. Se necesita crear un directorio denominado `MoodleBigData` en el directorio `user`. Dentro de él se deben crear los siguientes subdirectorios:

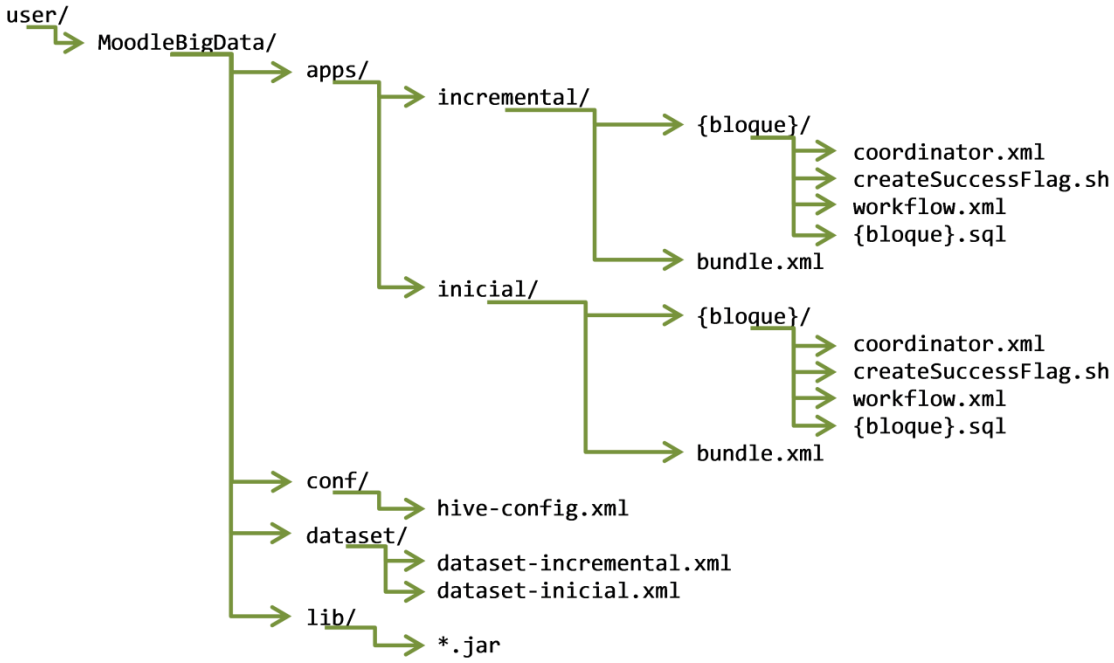


Figura 95. Estructuración de directorios necesaria en *HDFS*

- `apps`: con el contenido de la carpeta *apps* proporcionada en el código fuente.
- `conf`: con los ficheros de configuración, en este caso, tan solo `hive-config.xml` (ver sección 4.8.2). Es recomendable tomar los ficheros de configuración de la propia MV instalada, ya que contendrán la configuración que le corresponde a ella.
- `dataset`: con el contenido del directorio *dataset* del código fuente
- `lib`: con el contenido del directorio *lib* del código fuente.

Hue proporciona una opción de subir un fichero *ZIP* a *HDFS* a la vez que lo descomprime, es decir, si comprimimos los cuatro directorios indicados anteriormente en un fichero *ZIP* denominado *MoodleBigData* y lo subimos por medio de esta opción, en el directorio `user` de *HDFS*, se descomprime la estructura de directorios comprimida en el directorio `MoodleBigData` dentro de `user`.

La ejecución de las aplicaciones se encargará de crear los siguientes directorios o ficheros:

- `/user/MachineLearning`: directorio y estructura adicional para el almacenamiento de los resultados de los algoritmos de *Machine Learning* (ver sección 4.6.6) antes de ser procesados para guardar la información en *Hive*.
- `/user/MoodleBigData/flags`: directorio para el almacenamiento de los *flags* de completitud generados por los bloques a ejecutar en las aplicaciones.
- `/user/MoodleBigData/fecha`: fichero donde se guarda la fecha de la ejecución de la aplicación incremental, para su uso posterior por los distintos bloques.

4.5. Origen de datos

En este apartado trataremos con el origen de datos, es decir, la base de datos en la que se almacena la información de *Moodle*. En nuestro caso esta será una base de datos *MySQL*. Sobre el origen de datos trataremos dos aspectos: tablas *Moodle* que vamos a utilizar y cómo hemos escogido transferirlas a *Hadoop*.

4.5.1. Tablas *Moodle*

A continuación, procedemos a explicar las tablas *Moodle* utilizadas en el trabajo. Para cada una de ellas, indicaremos qué información guardan o qué información nos es útil de ellas, y los campos necesarios para nuestro trabajo (ver Tablas 27-67). Por simplicidad, las tablas se enumeran por orden alfabético. La información sobre la definición de los distintos módulos *Moodle* se pueden encontrar en [204]. El uso concreto de estas tablas, así como la relación entre todas ellas, se detalla en el código fuente del trabajo.

mdl_assign		
información sobre los *Assign* (tareas) definidos en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de *assign*
course	bigint	identificador de curso en el que se ha publicado

name	varchar	nombre del *assign*
duedate	bigint	fecha de entrega máxima del *assign*
teamsubmission	tinyint	indica si la entrega del *assign* es individual (0) o en grupo (1)

Tabla 27. Datos origen: Tabla mdl_assign

mdl_assign_submission		
información sobre las entregas de los usuarios a los *assign*		
campo	**tipo**	**descripción**
id	bigint	identificador de entrega
assignment	bigint	identificador de *assign* al que se realiza la entrega
userid	bigint	identificador de usuario que realiza la entrega
timemodified	bigint	fecha de modificación de la entrega, en formato *timestamp*
status	varchar	estado de la entrega, principalmente, entrega (‘*submitted*’) o en borrador (‘*draft*’)

Tabla 28. Datos origen: Tabla mdl_assign_submission

mdl_assignment		
anteriormente la información de *assign* se almacenaba como *assignments*		
campo	**tipo**	**descripción**
id	bigint	identificador de *assignment*
course	bigint	curso en el que está publicado
name	varchar	nombre del *assignment*

Tabla 29. Datos origen: Tabla mdl_assignment

mdl_book		
información sobre los libros publicados en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de libro
course	bigint	curso en el que está publicado
name	varchar	nombre del libro

Tabla 30. Datos origen: Tabla mdl_book

mdl_chat		
información sobre los chats definidos en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de chat
course	bigint	curso en el que está publicado
name	varchar	nombre del chat

Tabla 31. Datos origen: Tabla mdl_chat

mdl_choice		
información sobre los *Choice’s* definidos en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de *choice*
course	bigint	curso en el que está publicado
name	varchar	nombre del *choice*

Tabla 32. Datos origen: Tabla mdl_choice

mdl_config		
información sobre configuración de *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de elemento de configuración
name	varchar	nombre del elemento de configuración
value	longtext	valor del elemento de configuración

Tabla 33. Datos origen: Tabla mdl_config

mdl_context		
información sobre los contextos definidos en *Moodle*. En nuestro trabajo, nos interesa que los contextos van a definir cómo actúa cada usuario en cada curso		
campo	**tipo**	**descripción**
id	bigint	identificador de contexto
instanceid	bigint	instancia a la que se refiere el contexto

Tabla 34. Datos origen: Tabla mdl_context

mdl_course		
información sobre los cursos definidos actualmente en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador del curso en *Moodle*
fullname	varchar	nombre completo del curso
idnumber	varchar	serie numérica que caracteriza al curso
timecreated	bigint	fecha de creación del curso, en formato *timestamp*
startdate	bigint	fecha de inicio del curso, en formato *timestamp*

Tabla 35. Datos origen: Tabla mdl_course

mdl_course_modules		
información sobre los módulos definidos en cada curso. Un módulo se puede considerar como un contenedor donde se publican elementos como recursos, *assign*, libros, etc.		
campo	**tipo**	**descripción**
id	bigint	identificador de módulo (*cmid*)
course	bigint	identificador del curso en que se ha definido
module	bigint	identificador numérico del tipo de módulo
instance	bigint	identificador del módulo dentro de su tipo de elemento
added	bigint	fecha de adición o creación del módulo al curso, en formato *timestamp*

Tabla 36. Datos origen: Tabla mdl_course_modules

mdl_data		
información sobre las bases de datos definidas en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de BD
course	bigint	curso en el que está publicada
name	varchar	nombre de la BD

Tabla 37. Datos origen: Tabla mdl_data

mdl_enrol		
información sobre los tipos de enrolamientos posibles en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de *enrol*
courseid	bigint	identificador del curso sobre el que se está haciendo el enrolamien-

		to. En caso de ser un curso META, define el curso padre
enrol	varchar	tipo de *enrol*, principalmente: '*self*', '*manual*', '*guest*' o '*meta*'
customint	bigint	en caso de ser un *enrol* de tipo 'meta', guarda el identificador de un curso hijo

Tabla 38. Datos origen: Tabla mdl_enrol

mdl_feedback		
información sobre elementos de retroalimentación publicados en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de *feedback*
course	bigint	curso en el que está publicado
name	varchar	nombre del *feedback*

Tabla 39. Datos origen: Tabla mdl_feedback

mdl_folder		
información sobre las carpetas de elementos publicadas en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de carpeta
course	bigint	curso en el que está publicado
name	varchar	nombre de la carpeta

Tabla 40. Datos origen: Tabla mdl_folder

mdl_forum		
información sobre los foros publicados en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de foro
course	bigint	identificador de curso en que se ha publicado
name	varchar	nombre del foro

Tabla 41. Datos origen: Tabla mdl_forum

mdl_forum_discussions		
información sobre las discusiones publicadas en los foros		
campo	**tipo**	**descripción**
id	bigint	identificador de discusión
course	bigint	identificador del curso en que está publicada
forum	bigint	identificador del foro al que pertenece
name	varchar	nombre de la discusión
firstpost	bigint	identificador del primer *post* de la discusión
userid	bigint	identificador del usuario que crea la discusión

Tabla 42. Datos origen: Tabla mdl_forum_discussions

mdl_forum_post		
información sobre los *post* publicados en las discusiones de los foros		
campo	**tipo**	**descripción**
id	bigint	identificador de *post*
discussion	bigint	identificador de la discusión a la que pertenece el post
created	bigint	fecha de creación del *post*, en formato *timestamp*

Tabla 43. Datos origen: Tabla mdl_forum_post

mdl_glossary		
información sobre los glosarios definidos en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de glosario
course	bigint	curso en el que está publicado
name	varchar	nombre del glosario

Tabla 44. Datos origen: Tabla mdl_glossary

mdl_grade_grades		
información sobre las calificaciones dadas a los usuarios sobre los correspondientes elementos evaluables		
campo	**tipo**	**descripción**
id	bigint	identificador de la calificación
itemid	bigint	identificador del elemento evaluado
userid	bigint	identificador del usuario evaluado
finalgrade	bigint	nota obtenida por el usuario en el elemento

Tabla 45. Datos origen: Tabla mdl_grade_grades

mdl_grade_items		
definición de los elementos evaluables		
campo	**tipo**	**descripción**
id	bigint	identificador del ítem evaluable
courseid	bigint	identificador del curso en que está definido
iteminstance	bigint	identificador del elemento evaluable, dentro de su tipo de elemento, que se define en el ítem. Por ejemplo, si es un *assign*, guarda el identificador de *assign* según mdl_assign
grademax	decimal	nota máxima obtenible en el ítem
grademin	decimal	nota mínima obtenible en el ítem
itemtype	varchar	tipo de ítem que es el elemento, principalmente: '*course*' si se evalúa el curso entero o '*mod*' si se evalúa algún módulo del curso
itemmodule	varchar	si el ítem es un módulo, indica que tipo de módulo, por ejemplo, '*assign*' o '*quiz*'

Tabla 46. Datos origen: Tabla mdl_grade_items

mdl_groups		
información sobre los grupos de trabajo definidos en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de grupo
courseid	bigint	identificador del curso al que pertenece el grupo

Tabla 47. Datos origen: Tabla mdl_groups

mdl_groups_members		
información sobre la pertenencia de usuarios a grupos de trabajo		
campo	**tipo**	**descripción**
id	bigint	identificador de pertenencia
groupid	bigint	identificador de grupo al que pertenece el usuario
userid	bigint	identificador de usuario

Tabla 48. Datos origen: Tabla mdl_groups_members

mdl_imscp		
información sobre IMSCP definidos en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de IMSCP
course	bigint	curso en el que está publicado
name	varchar	nombre del IMSCP

Tabla 49. Datos origen: Tabla mdl_imscp

mdl_label		
información sobre etiquetas definidas en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de etiqueta
course	bigint	curso en el que está publicado
name	varchar	nombre de la etiqueta

Tabla 50. Datos origen: Tabla mdl_label

mdl_lesson		
información sobre lecciones definidas en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de lección
course	bigint	curso en el que está publicado
name	varchar	nombre de la lección

Tabla 51. Datos origen: Tabla mdl_lesson

mdl_log		
mantiene registros de la actividad realizada sobre *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador del registro
time	bigint	fecha del registro, en formato *timestamp*
userid	bigint	identificador del usuario que genera el registro
ip	varchar	dirección IP desde la que se realiza el registro
course	bigint	identificador del curso dónde se realiza el registro
module	varchar	tipo de módulo sobre el que se realiza el registro
cmid	bigint	identificador del módulo
action	varchar	acción llevada a cabo
url	varchar	*url* de la acción
info	varchar	información adicional

Tabla 52. Datos origen: Tabla mdl_log

mdl_lti		
información sobre recursos externos (LTI) definidos en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de LTI
course	bigint	curso en el que está publicado
name	varchar	nombre del LTI

Tabla 53. Datos origen: Tabla mdl_lti

mdl_modules		
definición de los distintos módulos publicables en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador numérico del tipo de módulo
name	varchar	nombre identificativo del tipo de módulo

Tabla 54. Datos origen: Tabla mdl_modules

mdl_page		
información sobre páginas definidas en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de página
course	bigint	curso en el que está publicado
name	varchar	nombre de la página

Tabla 55. Datos origen: Tabla mdl_page

mdl_quest		
información sobre *Questournaments* definidos en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de *quest*
course	bigint	curso en el que está publicado
name	varchar	nombre del *quest*

Tabla 56. Datos origen: Tabla mdl_quest

mdl_quiz		
información de *Quiz* (cuestionarios) definidos en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de *quiz*
course	bigint	identificador del curso en el que se ha definido el *quiz*
name	varchar	nombre del *quiz*

Tabla 57. Datos origen: Tabla mdl_quiz

mdl_quiz_attempts		
información sobre los intentos de realización de los *quiz*		
campo	**tipo**	**descripción**
id	bigint	identificador del intento
quiz	bigint	identificador de *quiz* sobre el que se realiza el intento
userid	bigint	identificador del usuario que realiza el intento
timestart	bigint	fecha de comienzo del intento, en formato *timestamp*
timefinish	bigint	fecha de fin del intento, en formato *timestamp*
state	varchar	estado del intento, principalmente: finalizado (‘*finished*’) o en progreso (‘*in_progress*’)

Tabla 58. Datos origen: Tabla mdl_quiz_attempts

mdl_resource		
información sobre recursos publicados en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de recurso
course	bigint	curso en el que está publicado
name	varchar	nombre del recurso

Tabla 59. Datos origen: Tabla mdl_resource

mdl_role_assignments		
define el rol con el que cada usuario accede a cada curso		
campo	**tipo**	**descripción**
id	bigint	identificador de asignación de rol
roleid	bigint	identificador de rol asignado
contextid	bigint	identificador del contexto donde se obtiene ese rol
userid	bigint	identificador del usuario que obtiene ese rol en ese contexto

Tabla 60. Datos origen: Tabla mdl_role_assingments

mdl_scorm		
información sobre *Scorm* publicados en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de *scorm*
course	bigint	curso en el que está publicado
name	varchar	nombre del *scorm*

Tabla 61. Datos origen: Tabla mdl_scorm

mdl_survey		
información de *Survey* publicados en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de *survey*
course	bigint	curso en el que está publicado
name	varchar	nombre del *survey*

Tabla 62. Datos origen: Tabla mdl_survey

mdl_url		
información de *URLs* definidas en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de *url*
course	bigint	curso en el que está publicado
name	varchar	nombre del *url*

Tabla 63. Datos origen: Tabla mdl_url

mdl_user		
información sobre los usuarios definidos en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador del usuario
idnumber	varchar	identificador univoco de usuario que le caracteriza más allá del identificador anterior

Tabla 64. Datos origen: Tabla mdl_user

mdl_user_enrolments		
define los enrolamientos de los usuarios en los cursos, es decir, a qué cursos puede acceder cada usuario		
campo	**tipo**	**descripción**
id	bigint	identificador del enrolamiento
enrolid	bigint	identificador del *enrol*, es decir, de a donde se enrola el usuario
userid	bigint	identificador del usuario que se enrola
timestart	bigint	fecha de comienzo del enrolamiento, en formato *timestamp*

Tabla 65. Datos origen: Tabla mdl_user_enrolments

mdl_wiki		
información sobre *Wikis* publicadas en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de la *wiki*
course	bigint	curso en el que está publicado
name	varchar	nombre de la *wiki*

Tabla 66. Datos origen: Tabla mdl_wiki

mdl_workshop		
información sobre *Workshops* publicados en *Moodle*		
campo	**tipo**	**descripción**
id	bigint	identificador de *workshop*
course	bigint	curso en el que está publicado
name	varchar	nombre del *workshop*

Tabla 67. Datos origen: Tabla mdl_workshop

4.5.2. Transferencia mediante *Sqoop*

Una vez analizadas las tablas de *Moodle* que necesitamos para nuestro trabajo, nos planteamos cómo importarlas, mediante *Sqoop,* a nuestro servidor *BigData*. Como ya se habrá comprobado en el apartado 4.2, las vamos a transferir a *Hive* ya que será este el servicio que utilicemos para analizar y procesar su contenido. Tras analizar las distintas alternativas indicadas anteriormente, se fueron descartando distintas opciones.

En primer lugar, se puede plantear una simple carga de todas las tablas de la base de datos de *Moodle*. En principio, todas las tablas cumplen la condición de poseer un campo de clave única, por lo que no habría problema. Esta opción se descarta ya que en *Moodle* no solamente tenemos las tablas que necesitamos sino otras tantas (aproximadamente las tablas que necesitamos son el 25% del total de tablas), lo que implica que estaríamos transfiriendo mucha información que no vamos a utilizar, por lo que nos decantamos por la importación de una única tabla, repetida para cada una de las tablas que necesitamos.

Por otro lado, nos planteamos usar una importación incremental para las sucesivas transferencias de las tablas a *Hadoop*, pero también fue descartada. Una importación incremental en modo *APPEND* sería prácticamente imposible de usar, ya que en las tablas *Moodle* no podemos garantizar que los registros ya transferidos no se actualicen, es más, sería lo más normal que se actualizasen. Existirían un par de casos en los que sí podríamos optar por esta opción. Entonces, lo lógico sería usar el modo *LASTMODIFIED*, consultando marcas temporales de modificación. En este sentido, existen una serie de tablas en las que sí podríamos tener un campo que se pudiera considerar como tiempo de modificación, pero tampoco lo encontraríamos en todos los casos. Además, *Sqoop* no lo plantea en su documentación, pero tras probar este último tipo de importación incremental, se ha comprobado que la herramienta no sustituye el registro antiguo por el nuevo, sino que simplemente lo añade, por lo que necesitaríamos realizar la sustitución manualmente con un *script* en *Hive*, por lo que las cosas se complican aún más.

A mayores, para realizar cualquier importación incremental necesitaríamos indicar un último valor, que tendríamos que obtener de *Hive* (mediante una consulta de último valor o valor máximo) y después pasárselo al comando *Sqoop*, algo bastante complicado de automatizar.

Dado todo esto, se ha optado por la opción más sencilla, aunque probablaemente no óptima: cada vez que se necesite recargar el valor de una tabla en *Hive* dado su origen externo en *Moodle*, se borrará la tabla y se volverá a crear. Antes de realizar la importación *Sqoop*, borraremos todas las tablas, mediante un script en *Hive*, y después las volveremos a transferir desde cero.

Antes de seguir describiendo el proceso y para entender el por qué, planteamos otra cuestión de *Sqoop*: las tablas transferidas se almacenan en formato texto plano, y según el manual de *Sqoop* no es posible hacerlo en otro formato. Esto nos va a obligar a hacer un procesamiento de estas tablas posterior a su importación a *Hive* y previo a su procesado *BigData*. Para optimizar el manejo de tablas en *Hive*, se recomienda usar el formato *ORC* (apartado 4.3.1) para almacenar la información, además, añadiremos opciones de *bucketing* y particionado a las tablas (apartado 4.3.6), en función de cómo se vayan a usar después, para optimizar la ejecución de consultas *Hive*.

Vamos a considerar tres grupos de tablas:

- `mdl_user`.
- *Conjunto* I formado por: `mdl_assign`, `mdl_assign_submission`, `mdl_config`, `mdl_context`, `mdl_course`, `mdl_course_modules`, `mdl_forum`, `mdl_forum_discussions`, `mdl_forum_posts`, `mdl_grade_grades`, `mdl_grade_items`, `mdl_groups`, `mdl_groups_member`, `mdl_log`, `mdl_modules`, `mdl_quiz`, mdl_quiz_attempts, `mdl_role_assignments`, `mdl_user_enrolments`.
- Conjunto II formado por: `mdl_feedback`, `mdl_chat`, `mdl_assignment`, `mdl_book`, `mdl_data`, `mdl_folder`, `mdl_glossary`, `mdl_imscp`, `mdl_label`, `mdl_lesson`, `mdl_lti`, `mdl_page`, `mdl_quest`, `mdl_resource`, `mdl_scorm`, `mdl_survey`, `mdl_url`, `mdl_wiki`, `mdl_workshop`.

El primer conjunto de tablas va a pasar por este procesado, por ello, se transferirán a *Hive* bajo el nombre `nombre_original_txt`, por ejemplo, la tabla `mdl_assign` se transferirá bajo el nombre `mdl_assign_txt`, para indicar que es la versión en texto plano que debe ser procesada. El procesamiento concreto que se haya escogido generará la tabla `mdl_assign`, que será la tabla que se utilice para el análisis *BigData*. El segundo conjunto de tablas no va a necesitar de procesado ya que estas tablas solo se van a usar para coger su información y, juntando los datos de todas ellas, generar una única tabla, por lo tanto ya directamente cogeremos la información de la tabla en texto plano y lo guardaremos realizando el procesamiento, que en este caso será guardar en formato *ORC*. Así, las tablas de este segundo conjunto, serán transferidas directamente con el mismo nombre.

Fuera de estos dos conjuntos se maneja `mdl_user`, ya que necesitamos que se transfiera antes que todas ellas. A partir de los usuarios definidos en ella, generaremos una tabla en la que para cada usuario guardaremos sus campos `id` e `idnumber`. El campo `id` identifica al usuario en *Moodle* mediante un número que se va asignando de forma secuencial a los usuarios según se van añadiendo, pero no es un buen identificador genérico de cada usuario, ya que puede ocurrir que pasado un cierto tiempo, por ejemplo, con cada año académico, los identificadores se reasignen y ya no se refieran al mismo usuario, por ello, debemos encontrar un identificador mucho más personal para cada usuario. Analizando la forma de funcionamiento del *Moodle* de la universidad, parece ser que en el campo `idnumber` se guarda un identificador de la forma `eDNI-usuario`, por lo tanto parece buena idea tomar este campo como identificador inequívoco de usuario. Así, como toda la información de *Moodle* se referencia con el identificador de usuario, el preprocesado anteriormente comentado, en aquellos casos

en que sea necesario, va a sustituir el identificador *Moodle* del usuario, por el campo `idnumber` que le corresponde. Para ello, se accederá a una tabla creada a partir de `mdl_user` donde se guarda para cada identificador *Moodle*, el `idnumber` que le corresponde actualmente. Es de suponer, que todas las estadísticas para un mismo identificador *Moodle* de usuario, se refieren a un mismo usuario físico y con el mismo DNI. De esta forma, ya tenemos la información de *Moodle* referida con el nuevo identificador.

La transferencia se realizará mediante comandos del siguiente tipo:

```
import --connect jdbc:{tipoDB}://{host}:{port}/{db} --username {user} --
password {pass} --hive-import --hive-overwrite --hive-database moodle2 --
hive-drop-import-delims  --table mdl_assign --hive-table mdl_assign_txt
```

Como se puede comprobar del ejemplo anterior, la copia de los datos de *Moodle* se guardarán en una base de datos denominada `moodle2`, esto es debido a que la plataforma utilizada como referencia para la realización de este trabajo es una plataforma *Moodle* versión 2, cuyo nombre es `moodle2`.

En cuanto a las opciones, aunque las tablas se recrean con cada análisis, se ha añadido la opción `--hive-overwrite`, por si en un futuro se quiere cambiar la forma de realizar la importación, tener más facilidad de cambio de todas las sentencias de *Sqoop*. También se ha añadido la opción `--hive-drop-import-delims`. Cuando la información se guarda en texto plano, se le debe decir a Hive cómo debe delimitar la información para saber a qué registro y dentro de cada registro a qué columna pertenece, si no se especifica, *Hive* tiene sus delimitadores por defecto, no nos importan los delimitadores que se usen, pero sí que estos coincidan con caracteres utilizamos para dar valor al campo. Esta opción los elimina, para evitar problemas de lectura de los datos almacenados en los ficheros. Fundamentalmente esta opción elimina los saltos de línea `\n`, los retornos de carro `\r` y el carácter `\01`.

Recordar que a continuación de la transferencia individual de cada tabla, se añade una etapa de procesado según se ha indicado anteriormente, implementada por medio de *Hive*.

4.6. Análisis *BigData*

En este apartado se analiza la parte central del trabajo realizado: la utilización de *Hive* para generar, a partir de los datos de *Moodle*, un análisis *BigData* de la actividad. El contenido se ha dividido en seis apartados:

- En el apartado 4.6.1 trataremos la estrategia de trabajo seguida para desarrollar el análisis a modo de introducción al resto de apartados.
- En el apartado 4.6.2 indicaremos el preprocesado al que hemos sometido a cierta información de *Moodle* para guardarla de una forma más conveniente para facilitar el análisis.
- En el apartado 4.6.3 desarrollaremos la parte del análisis relacionada con las estadísticas de acceso.
- En el apartado 4.6.4 se detalla la parte del análisis que nos ha proporcionado información sobre los resultados académicos de los alumnos.

- En el apartado 4.6.5 se obtienen métricas que muestran la influencia de la actividad en *Moodle* sobre los resultados de los alumnos.
- Por último, en el apartado 4.6.6, se trata con algoritmos de *Machine Learning* para encontrar resultados más avanzados.

4.6.1. Estrategia de trabajo

Procedemos a explicar la estrategia de trabajo. En primer lugar, anteriormente se ha comentado que la información que proviene de *Moodle* se guardará en una base de datos *Hive* denominada `moodle2`. Todos los resultados obtenidos se guardan en otra base de datos *Hive* denominada `moodle2_bigdata_results`. La creación de estas bases de datos es la única operación que el usuario de esta aplicación debe realizar manualmente. Esta creación es idéntica a cualquier otra base de datos *SQL*.

La solución planteada en este trabajo se dividirá en dos partes: un primer **análisis inicial** y una serie de **análisis incrementales**. El análisis inicial consistirá en generar resultados iniciales (para las distintitas métricas planteadas) partiendo de cero, sin ningún resultado anterior, y por lo tanto, sin tener que hacer suposiciones previas. Obviamente, este primer análisis puede que tenga que hacer frente a grandes cantidades de datos y requiera de un gran tiempo de procesamiento.

Una vez completado el análisis inicial y comprobado que se han captado los resultados requeridos, se realizarán una serie de análisis incrementales. En concreto, cada día a las 00:00 se realizará un análisis sobre qué ha ocurrido justo el día anterior y se añadirá a los resultados anteriores. A partir de este histórico de registros, se actualizarán los registros absolutos de las métricas correspondientes. Es decir, nos estamos construyendo una versión más ordenada (según nuestras necesidades) de los registros de *Moodle*, para poder seguir utilizándolos aunque estos se borren de *Moodle*. A partir de este registro simulado, se recalculan las correspondientes métricas, para generar los resultados del análisis correspondiente. Como se ha indicado en el apartado 4.3.6, para ciertas métricas, se guardarán los resultados obtenidos cada día, para mantener un histórico que en un futuro pueda permitir realizar un análisis de la evolución.

En la práctica, para manejar situaciones en que un día no se haya podido realizar un análisis, se analizarán solo los días entre el último día que sí se analizó y el anterior al actual. En un análisis nunca se incluye el análisis del día actual, ya que esté todavía no se ha completado y puede seguir produciéndose información relativa a él.

En resumen: el día 0 hacemos el análisis inicial con todos los datos que tengamos hasta el día anterior (el día 0 aún no se ha completado) inicializando todas las tablas de resultados y demás. La noche del día 0 al día 1 (más allá de las 00:00) se analiza qué ha ocurrido el día 0, y se añaden estos datos indicando que son del día 0. La noche del día 1 al día 2 se analiza qué ha ocurrido el día 1, la noche del día 2 al día 3 se analiza qué ha ocurrido el día 2, etc. Si la noche del día 1 al día 2 no se pudo realizar el análisis y no se generaron datos, la noche del día 2 al día 3 se analizará qué ocurrió los días 1 y 2. Para todo ello, nos fijamos en el último día para el que hay entradas en los resultados.

Un punto todavía a analizar es qué pasa cuando se generan parte de los datos, es decir, hay un fallo en mitad del análisis y no se copian todos los datos requeridos para el día X. Habría que comprobar cuántos datos deberían haberse generado para ese día y si la cantidad es infe-

rior, borrar los que existan (o simplemente crear de nuevo la tabla sin seleccionarlos) e iniciar el análisis a partir del último día que sí se completó. Por lo tanto, una buena práctica previa, será comprobar que se han generado todos los datos. Se podría hacer como última etapa del análisis, pero si este falla a mitad no se garantiza que esta última etapa se ejecute. Así, se optará por añadirlo como primera etapa del análisis siguiente, tras lo cual, la última fecha ya no es la incompleta y se hace de nuevo todo el análisis.

Se debe prestar atención en que los análisis iniciales están pensados para ser realizados comprobando que generan resultados. Las consultas están probadas, por lo que funcionan correctamente, pero sería adecuado controlar que se guardan los resultados. Por ejemplo, pueden no generarse resultados si no se encuentran datos en la base de datos de *Moodle*. Una vez que ya se hayan generado datos se puede programar el uso de las consultas incrementales. Esto es debido a que, para realizar las consultas incrementales se busca información del análisis anterior, así que en primera instancia se buscan datos del análisis inicial, por ejemplo, la última fecha para la que se generaron datos. Se ha intentado que las consultas incrementales puedan llegar a generar los datos iniciales si el análisis inicial no obtuvo resultados.

Indicar que, cualquiera de los resultados se obtiene tras varias etapas de procesado que generan una serie de tablas que se han denominado temporales. Se ha optado por mantener estas tablas temporales tras su utilización en la obtención del resultado final, por si hubiera algún error de ejecución de las sentencias, poder tener más posibilidades para recuperar datos.

Tal y como se infiere del índice del apartado enumerado anteriormente, vamos a distinguir cinco bloques de tablas:

- Información directa de *Moodle* pero expresada de forma más conveniente.
- Información sobre estadísticas de acceso, entre las que distinguiremos, las tablas con la versión histórica del registro de *Moodle* a partir de las cuáles se recalculan las demás, así como otra serie de resultados intermedios necesarios para obtener más cómodamente los resultados finales.
- Información sobre resultados académicos, obtenidos directamente de *Moodle*.
- Análisis estadístico de la relación entre los dos bloques anteriores.
- Información relacionada con *Machine Learning*.

Para facilitar el trabajo a realizar así como llevar un mayor control, hemos definidos tres conjuntos de tablas con las que ordenar e indicar qué datos queremos obtener.

El punto clave de todo el trabajo, está en una simple tabla en la que almacenaremos el **último día** en que supimos que cada uno de los registros históricos de la actividad en *Moodle* se completó con éxito, para poder analizar si ha habido algo erróneo y se debe volver a realizar. Además, guardaremos la fecha del análisis que se está actualmente realizando (qué será la fecha actual del ordenador menos un día), para poder utilizar este campo como variable en todo el análisis, y si este se extiende por más de un día, se siga usando el mismo valor. Su estructura se muestra, junto con su valor inicial, en la Tabla 68.

El análisis inicial, toma estos valores. Tras completar la etapa de análisis de la tabla de cursos, actualiza al día del análisis la entrada para cursos. Cada análisis incremental, tras realizar la comprobación del último análisis, actualiza al último día correcto las restantes entradas. Las

operaciones relativas a esta tabla se han denominado `UltimoDia`. Esta comprobación se basa en:

- Calcular, para cada tabla a analizar y a partir de la plantilla utilizada en el análisis anterior, la cantidad de datos que se debían haber guardado por día.
- Calcular, para cada día, los datos que se guardaron en cada tabla.
- Seleccionar aquellos días en que ambas métricas no coinciden y quedarse con el mínimo de todos ellos.
- Borrar toda la información cuya fecha sea mayor o igual a ese día obtenido.
- Actualizar los campos de `UltimoDia` que correspondan, para guardar la mayor fecha obtenida de cada tabla.

moodle2_bigdata_results.ultimo_dia		
Campo (tipo)		**Descripción**
análisis (string)	**dia (date (YYYY-MM-dd))**	
cursos	**1970-01-01**	fecha de la última vez que se comprobó el contenido de `mdl_course` y se actualizó el contenido de `cursos_actuales` y `cursos_historico`
acceso-curso-usuario	**1970-01-01**	fecha del último día en que se comprobó que se completó correctamente la tabla de resultados `EstDCU`
acceso-modulos	**1970-01-01**	fecha del último día en que se comprobó que se completó correctamente la tabla de resultados `EstDCUM`
acceso-foros	**1970-01-01**	fecha del último día en que se comprobó que se completó correctamente la tabla de resultados `EstDCUF`
acceso-discusiones	**1970-01-01**	fecha del último día en que se comprobó que se completó correctamente la tabla de resultados `EstDCUFD`
acceso-modulos-foros	**1970-01-01**	mínimo de los campos `accesos-modulos` y `accesos-foros`
dia-analisis	**{día actual-1}**	fecha correspondiente al día del análisis

Tabla 68. Estructura y contenido inicial de la tabla `ultimo_dia`

En segundo lugar, distinguimos las **tablas de historial**, aquellas que nos permitirán mantener toda la información de construcción de cursos en *Moodle*, aunque se borre de su origen. Tendremos las siguientes tablas de historial, analizadas en el apartado 4.6.2:

- Cursos.
- Matriculaciones.
- Módulos.
- Foros.

- Discusiones.
- *Assign*.
- *Quiz*.
- Grupos.

El tercer conjunto de tablas que vamos a definir se han denominado **plantillas**. Con ellas, vamos a registrar todas las combinaciones, de los campos o valores que corresponda, que necesitamos para definir qué información queremos registrar. La estrategia es partir de estas plantillas e ir añadiendo columnas a la derecha con nuevas métricas. El motivo de usarlas está en poder obtener información completa, es decir, realizando pruebas se comprobó que al calcular distintas métricas, a partir de resultados previos o directamente de la información de *Moodle*, no se tenía por qué obtener valores para todas las combinaciones que necesitamos, por ejemplo:

- Cuando calculábamos el número de accesos a cada curso por usuario y día mediante las opciones de agrupamiento (`GROUP BY`) de *HiveQL* a partir de `mdl_log`, solo obteníamos aquellas combinaciones curso – usuario – día para las cuáles se habían producido accesos. De esta forma, si un usuario no accedía un día a un curso no se generaba datos para esa combinación curso – usuario – día.

- Obtenemos la calificación que ha obtenido el usuario en una tarea evaluable o en el curso en total, a partir de `mdl_grade_grades`. Si el usuario no ha obtenido calificación, analizando `mdl_grade_grades` no obtendremos ninguna entrada para su relación curso – usuario, que en este caso tendría que tener algo de la forma “Sin calificar”.

Parece una práctica mucho más coherente poder registrar los no accesos mediante un 0, las no calificaciones mediante un “Sin calificar” o el resto de métricas según corresponda, a no tener información. Esto nos lo permite el uso de las plantillas junto con la sentencia `LEFT OUTER JOIN` de *HiveQL* (ya explicada en el apartado 4.3.5). Mediante esta sentencia trataremos de unir la plantilla que corresponda con las distintas sentencias *HiveQL* que calculan distintas métricas y así poder registrar todas las combinaciones deseadas y aquellas para las qué no se encontró datos, guardar lo que la lógica pida. Realmente, cuando no se encuentran datos se deja la casilla en blanco o sin valor, pero con la función `COALESCE` le podemos indicar qué queremos que ponga en lugar de dejar la casilla sin valor, como se indicó en el apartado 4.3.5. Además, si en una misma tabla queremos guardar varias métricas, deberemos tener todos los valores, nulos o no, para que las columnas cuadren, por lo que el uso de estas plantillas gana enorme interés.

Otra de las ventajas de las plantillas es que le indicamos a las sentencias *HiveQL* qué queremos que busquen. Actúan como a modo de filtro de los datos analizados ya que solo se guardarán los datos que coincidan con alguna entrada de la plantilla. De esta forma, nos podemos olvidar de situaciones como la existencia de entradas incompletas, por ejemplo, en las que no se haya guardado qué usuario realizó el acceso, cuando queramos centrar el análisis en un rango temporal no deberemos filtrar la sentencia *HiveQL* que calcula la métrica; o si una calificación dada no se corresponde a ningún usuario registrado no tenemos que eliminarla; simplemente utilizamos una plantilla que solo comprenda las combinaciones deseadas, o si queremos centrar el análisis en un curso, cogemos solo las entradas de la plantilla relacionadas con ese curso. En todos los casos podemos usar la misma sentencia *HiveQL*, será la plantilla la que se encargue de filtrar todos aquellos resultados que no queremos.

De entre estas plantillas podemos distinguir dos grupos:

- Temporales: almacenan combinaciones repetidas a lo largo de un rango de tiempo. Estas plantillas se generan por medio de la estructura `LATERAL VIEW` y la *UDF* `DateRange` explicadas en los apartados 4.3.5 y 4.3.4, respectivamente. Son:
    - Plantilla día – curso – usuario, referida como bloque `PlantillaDCU`.
    - Plantilla día – curso – usuario – módulo, referida como bloque `PlantillaDCUM`.
    - Plantilla día – curso – usuario – foro, referida como bloque `PlantillaDCUF`.
    - Plantilla día – curso – usuario – foro – discusión, referida como bloque `PlantillaDCUFD`.
- Simples: almacenan combinaciones de elementos dentro de un mismo curso, sin rango de tiempo. Estas plantillas se generan mediante un cruce de tablas con `CROSS JOIN` (apartado 4.3.5). Son:
    - Curso – módulo – usuario, referida como `CursoModuloUsuario`.
    - Curso – foro – usuario, referida como `CursoForoUsuario`.
    - Curso – *assign* – usuario – grupo, referida como `CursoAssignUsuario`.
    - Curso – *quiz* – usuario, referida como `CursoQuizUsuario`.

La **plantilla día – curso – usuario** (`PlantillaDCU`, tabla `moodle2_bigdata_results.plantilla_dia_curso_usuario`) guarda todas las relaciones día – curso – usuario, es decir, para cada pareja curso – usuario, obtenida de la tabla de matriculaciones, se crea una lista de tripletas día – curso – usuario, donde día irá:

- Desde el valor `inicio` de la matriculación hasta el día anterior al actual, si la matriculación está activa (`activo = 'ACTIVO'`).
- Desde el valor `inicio` de la matriculación al final de la matriculación, si está desactivada (`activo = 'NO ACTIVO'`).

En caso de que la matriculación esté activada, solo se calculan datos hasta el día anterior al de realización del análisis, porque en cada análisis, el día actual no se incluye.

De esta forma, se están registrando solo los instantes de tiempo en que es posible que el usuario pueda entrar a ese curso y así registrar todas las métricas.

moodle2_bigdata_results.matriculaciones					
curso	usuario	inicio	fin	status	activo
0003-2015//0004-2015	2	2015-10-10	None	PROFESOR	ACTIVO
0001-2015	1	2015-10-01	None	PROFESOR	ACTIVO
0001-2015	3	2015-10-01	None	ALUMNO	ACTIVO
0003-2015//0004-2015	1	2015-10-10	2015-11-01	ALUMNO	NO ACTIVO
0001-2015	4	2015-11-01	None	ALUMNO	ACTIVO

Tabla 69. Ejemplo de tabla de matriculaciones I

Supongamos que tenemos la tabla de matriculaciones de la Tabla 69 y que realizamos el análisis el día 2015-11-05 (registrado en `UltimoDia`). La plantilla resultante será (por simplicidad no se muestran todas las entradas) la mostrada en la Tabla 70.

moodle2_bigdata_results.plantilla_dia_curso_usuario		
dia	curso	usuario
2015-10-10	0003-2015//0004-2015	1
…		
2015-11-01	0003-2015//0004-2015	1
2015-10-10	0003-2015//0004-2015	2
…		
2015-11-04	0003-2015//0004-2015	2
2015-10-01	0001-2015	1
…		
2015-11-04	0001-2015	1
2015-10-01	0001-2015	3
…		
2015-11-04	0001-2015	3
2015-11-01	0001-2015	4
…		
2015-11-04	0001-2015	4

Tabla 70. Plantilla día – curso – usuario a partir de Tabla 69

La **plantilla día – curso – usuario – módulo** (`PlantillaDCUM`, tabla `moodle2_bigdata_results.plantilla_dia_curso_usuario_modulo`) guarda todos los instantes de tiempo en que un usuario puede acceder a cada uno de los módulos de cada curso. El campo `dia` irá del máximo entre el inicio del módulo y el inicio de la matriculación (`max(inicio_modulo, inicio_matriculacion)`), hasta el mínimo entre el fin del módulo y el fin de la asociación (`min(fin_modulo, fin_matriculacion)`), donde:

- Si el módulo está activo, `fin_modulo = día actual – 1`.
- Si la matriculación está activa, `fin_matriculacion = día actual – 1`.

En caso de que un módulo y un usuario no tengan coincidencia temporal, no se generarán registros, ya que el cálculo de fechas anteriores dará lugar a una fecha de fin anterior a la de inicio, y la función `DateRange` no generará ningún rango de fechas.

moodle2_bigdata_results.cursos_modulos								
curso	modulo	identi-ficador	tipo	nombre	inicio	fin	inicio_timestamp	activo
0003-2015//0004-2015	1	1	resource	Recurso 1 del curso 2 modificado	2015-10-10	None	1444474800	ACTIVO
0001-2015	3	1	forum	Recurso 1 del curso 3	2015-10-15	None	1444906800	ACTIVO
0003-2015//0004-2015	2	1	resource	Recurso 2 del curso 2	2015-10-20	2015-11-01	1445338800	NO ACTIVO
0001-2015	4	1	assign	Recurso 2 del curso 3	2015-11-01	None	1446336000	ACTIVO

Tabla 71. Ejemplo de tabla de módulos I

Supongamos que tenemos las tablas de matriculaciones de la Tabla 69 y módulos de la Tabla 71 y que se realiza el análisis el día 2015-11-05; obtendremos el resultado de la Tabla 72 (por simplicidad no se muestran todas las entradas).

moodle2_bigdata_results.plantilla_dia_curso_usuario_modulo			
dia	curso	usuario	modulo
2015-10-10	0003-2015//0004-2015	1	1
…			
2015-11-01	0003-2015//0004-2015	1	1
2015-10-20	0003-2015//0004-2015	1	2
…			
2015-11-01	0003-2015//0004-2015	1	2
2015-10-10	0003-2015//0004-2015	2	1
…			
2015-11-04	0003-2015//0004-2015	2	1
2015-10-20	0003-2015//0004-2015	2	2
…			
2015-11-01	0003-2015//0004-2015	2	2
2015-10-15	0001-2015	1	3
…			
2015-11-04	0001-2015	1	3
2015-11-01	0001-2015	1	4
…			
2015-11-04	0001-2015	1	4
2015-10-15	0001-2015	3	3
…			
2015-11-04	0001-2015	3	3
2015-11-01	0001-2015	3	4
…			
2015-11-04	0001-2015	3	4
2015-11-01	0001-2015	4	3
…			
2015-11-04	0001-2015	4	3
2015-11-01	0001-2015	4	4
…			
2015-11-04	0001-2015	4	4

Tabla 72. Plantilla día – curso – usuario – modulo obtenida a partir de Tabla 69 y Tabla 71

La **plantilla día – curso – usuario – foro** (`PlantillaDCUF`, tabla: `moodle2_bigdata_results.plantilla_dia_curso_usuario_foro`) guarda todos los instantes de tiempo en que un usuario puede acceder a cada uno de los foros de cada curso. Se calcula a partir de la `PlantillaDCUM`, escogiendo las entradas para los módulos que son de tipo *forum*. Por ejemplo, a partir de la Tabla 72, obtendríamos la Tabla 73.

La **plantilla día – curso – usuario – foro – discusión** (`PlantillaDCUFD`, tabla: `moodle2_bigdata_results.plantilla_dia_curso_usuario_foro_discusion`) guarda todos los instantes de tiempo en que un usuario puede acceder a cada una de las discusiones de cada curso, detalladas por foro. El campo `dia` irá del máximo entre el inicio de la discusión y el inicio de la matriculación (`max(inicio_discusion, inicio_matriculacion)`), hasta el mínimo entre el fin de la discusión y el fin de la asociación (`min(fin_discusion, fin_matriculacion)`), donde:

- Si la discusión está activa, `fin_discusion = día actual – 1`.

- Si la matriculación está activa, `fin_matriculacion = día actual – 1`.

Su obtención es análoga a la `PlantillaDCUM`.

moodle2_bigdata_results.plantilla_dia_curso_usuario_foro			
dia	curso	usuario	foro
2015-10-15	0001-2015	1	3
…			
2015-11-04	0001-2015	1	3
2015-10-15	0001-2015	3	3
…			
2015-11-04	0001-2015	3	3
2015-11-01	0001-2015	4	3
…			
2015-11-04	0001-2015	4	3

Tabla 73. Plantilla día – curso – usuario – foro obtenida a partir de Tabla 72

En cualquiera de las plantillas, se pueden usar sentencias *HiveQL* que generan todos los datos. Obviamente estas plantillas crecen rápidamente con el tiempo. Se debe tener en cuenta que generar cualquiera de estas plantillas cuando la actividad en *Moodle* está bastante avanzada, llevará mucho tiempo. Por ello, en los análisis incrementales, se generará solo la parte de plantilla que se necesita, teniendo en cuenta desde qué día se debe realizar el análisis en base a cuál fue el último día analizado. Por ejemplo, supongamos que queremos generar la `PlantillaDCU` incremental. Para cada combinación curso – usuario solo querremos registrar los instantes de tiempo que no hayamos analizado anteriormente. Para ello, nos fijamos en la tabla `UltimoDia`, la cual en el registro `accesos-curso-usuario`, nos indica cual es la última fecha guardada en el análisis que depende de `PlantillaDCU`, y por tanto la fecha a partir de la cuál generar las combinaciones día – curso – usuario. Así, supongamos que tenemos de nuevo la situación de la Tabla 69, como fecha en `UltimoDia`, '2015-11-02' y fecha del análisis, '2015-11-05', se obtendría el resultado de la Tabla 74.

moodle2_bigdata_results.plantilla_dia_curso_usuario		
dia	curso	usuario
2015-11-03	0003-2015//0004-2015	2
2015-11-04	0003-2015//0004-2015	2
2015-11-03	0001-2015	1
2015-11-04	0001-2015	1
2015-11-03	0001-2015	3
2015-11-04	0001-2015	3
2015-11-03	0001-2015	4
2015-11-04	0001-2015	4

Tabla 74. Ejemplo plantilla día – curso – usuario incremental

De forma, similar se obtienen el resto de plantillas temporales incrementales.

Las plantillas simples, son más fáciles de generar, ya que solamente deberemos cruzar la información de dos tablas, basándose en el identificador de curso. La generación será siempre la misma en función de la información que en cada análisis tengan las tablas implicadas.

La **plantilla curso – módulo – usuario** (`CursoModuloUsuario`, tabla: `moodle2_bigdata_results.curso_modulo_usuario`) mantiene, para cada curso, todas las combinaciones entre los usuarios matriculados en el curso y los módulos publicados en él. Se está suponiendo que todos los usuarios pueden acceder a todos los módulos, por simplicidad, ya que el único problema que se genera son datos en exceso pero nunca pérdida de datos. Por

ejemplo, dada las matriculaciones de la Tabla 69 y los módulos de la Tabla 71, obtenemos el resultado de la Tabla 75.

moodle2_bigdata_results.curso_modulo_usuario		
curso	modulo	usuario
0001-2015	3	1
0001-2015	3	3
0001-2015	3	4
0001-2015	4	1
0001-2015	4	3
0001-2015	4	4
0003-2015//0004-2015	1	1
0003-2015//0004-2015	1	2
0003-2015//0004-2015	2	1
0003-2015//0004-2015	2	2

Tabla 75. Ejemplo plantilla curso – módulo – usuario

La **plantilla curso – foro – usuario** (`CursoForoUsuario`, tabla: `moodle2_bigdata_results.curso_foro_usuario`) mantiene, para cada curso, todas las combinaciones entre los usuarios matriculados en el curso y los foros publicados en él. Se obtiene de forma similar a la plantilla curso – módulo – usuario, a partir de la tabla de matriculaciones y de foros

La **plantilla curso – *assign* – usuario** (`CursoAssignUsuario`, tabla `moodle2_bigdata_results.curso_assign_usuario`) mantiene, para cada curso, todas las combinaciones entre los alumnos matriculados en él y los *assign* propuestos (actualmente, ver estructura de la tabla de *assign*) para que los alumnos los realicen, es decir, no genera combinaciones que incluyan profesores. Esto es debido a que esta plantilla se va a usar para completar una tabla de calificaciones de *assign*, obviamente los profesores no deberán formar parte de ella. Se obtiene de una forma similar a la plantilla curso – módulo – usuario, a partir de las tablas de matriculaciones y *assign*, pero añade una columna más de información, a cada combinación *assign* – usuario en cada curso, le añade el posible grupo de trabajo al que pertenece el usuario, ya que algunas entregas pueden ser en grupo y puede ser necesario no saber el usuario de la tarea sino el grupo. En caso de que el usuario no tenga definido ningún grupo, el campo se deja vacío. Esta información la obtiene de la tabla de grupos.

La **plantilla curso – *quiz* – usuario** (`CursoQuizUsuario`, tabla `moodle2_bigdata_results.curso_quiz_usuario`) mantiene, para cada curso, todas las combinaciones entre los alumnos matriculados en él y los *quiz* propuestos (actualmente, ver estructura de la tabla de *quiz*) para que los alumnos los completen, es decir, no genera combinaciones que incluyan profesores, por la misma razón que en la plantilla anterior. Se obtiene de forma similar a la plantilla curso – módulo – usuario, obteniendo la información de la tabla de matriculaciones y la tabla de *quiz*.

4.6.2. Preprocesado de la información de *Moodle*

A pesar de que *Moodle* nos da información sobre cómo se construyen los cursos, qué usuarios pueden acceder, qué información es accesible para ellos y demás, no está preparada para el análisis que queremos desarrollar. Principalmente, si un curso se borra de *Moodle*, es de esperar que se borre toda su información asociada, pero en nuestro estudio nos convendría mantener esa información, por ello necesitamos una etapa de preprocesado de la información de *Moodle*, principalmente por dos objetivos:

- Juntar bajo una misma tabla información que se encuentra repartida por varias tablas en *Moodle*.
- Mantener información histórica sobre cursos que ya no existen, es decir, se han borrado.

El conjunto de tablas que aquí se enumeran, son las que en el apartado anterior se han denominado tablas historial, dado su objetivo. Vamos a distinguir las siguientes:

- Cursos.
- Matriculaciones.
- Módulos.
- Foros.
- Discusiones.
- *Assign*.
- *Quiz*.
- Grupos.

Para mantener el **historial de cursos** (bloque `Cursos`, tablas: `moodle2_bigdata_results.cursos_actuales` y `moodle2_bigdata_results.cursos_historico`) tendremos dos tablas, una de ellas realmente solo guarda los cursos que actualmente están activos, considerando que están activos porque existen en la base de datos de *Moodle*. Por otro lado, tendremos la tabla de historial de cursos, propiamente dicha, en la que guardaremos la información administrativa de todos los cursos que, en algún momento, han sido analizados por la infraestructura *BigData*.

Como tenemos que manejar el supuesto de que los identificadores de curso se reutilicen o reasignen, los resultados de los análisis se guardarán con un identificador propio que daremos a los cursos y que denominaremos identificador *BigData*. Este no es más que el identificador *Sigma* o administrativo que tiene el curso. Si se revisa detenidamente la nomenclatura utilizada en la Universidad de Valladolid para dar nombre a sus cursos en la plataforma *Moodle*, estos siguen una estructura resultado de combinar el nombre de la asignatura con un código numérico, por ejemplo:

- Cursos únicos:

`ELECTRONICA DE COMUNICACIONES (1-211-244-43783-1-2012)`

Dónde:

  - `1`: identificador de campus.
  - `211`: identificador de facultad/escuela.
  - `244`: identificador de plan de estudios.
  - `43783`: identificador de asignatura.
  - `1`: identificador de grupo en la asignatura.

- `2012`: identificador de año académico. En realidad el año académico sería 2012 – 2013.

- Agrupación de varios cursos, lo que denominaremos CURSOS *META*:

`TEORÍA DE LA COMUNICACIÓN (META27491-2014)`

Dónde:

- `META27491`: identificador de la agrupación.
- `2014`: año académico. En realidad el año académico sería 2014 – 2015.

De estos ejemplos hemos concluido que cada combinación de estos parámetros es única, por lo que un buen identificador único global sería la combinación numérica indicada entre paréntesis en el nombre de cada curso. A esta combinación la vamos a denominar serie del curso. Una misma asignatura a lo largo de varios años académicos tendrá todos los campos iguales menos el último que indica el año, por lo que también podemos distinguir la misma asignatura pero en distintos años (y poder realizar análisis comparativos). Esta serie será nuestro identificador *BigData*. En el caso de la agrupación de cursos, la cosa se complica, ya que no podemos asegurar que el identificador de la agrupación se repita a lo largo de los años. Para distinguir cursos agrupados tendremos que recurrir al identificador *Sigma* de los cursos que forman la agrupación (cursos hijos). Por ejemplo, si un curso agrupa dos cursos, cuyos códigos *Sigma* son, respectivamente, `1-211-244-43783-1-2012` y `1-211-244-43783-2-2012`, su identificador *BigData* será: `1-211-244-43783-1-2012//1-211-244-43783-2-2012` (los cursos se ordenan alfabéticamente según su identificador).

Se ha comprobado, que este identificador *Sigma* se almacena bajo el campo denominado `idnumber` de la tabla `mdl_course`, por lo tanto, el único requerimiento para que funcione nuestra propuesta *BigData*, es que todos los cursos estén identificados unívocamente mediante este campo, con un identificador que permita diferenciar el curso a lo largo de los años, es decir, de la forma: `Identificador-de-curso-AÑO`, por lo tanto, son válidos cualquiera de los ejemplos anteriores, pero también nomenclaturas de la forma: `Curso1-2015`, `Matemáticas-2015`, `Economía-Grupo1-2015`, ya que mantiene la estructura, aunque no sea numérica. El único requisito para comparar un curso con el mismo curso en otros años, es que se llame igual, pero cambiando el año, es decir, `Curso1-2015` se podrá comparar con `Curso1-2013`, `Curso1-2014`, `Curso1-2016`, etc., ya que la parte que no comprende el año académico, es la misma.

Pero, los datos de *Moodle* vienen dados para el identificador *Moodle* y nosotros queremos guardarlos con nuestro identificador propio. Así, la tabla de cursos actuales tan solo guardará la correspondencia entre el identificador *Moodle* y el identificador *BigData* que llevaría asociado ese curso. Cuando estemos analizando, por ejemplo, `mdl_log`, mapearemos el identificador de curso *Moodle* al que se corresponda en nuestra tabla de cursos actuales. Por supuesto, aparece un margen de error, si estamos realizando el análisis en fechas próximas a la reutilización del identificador de curso, es posible que los resultados se encuadren en un curso equivocado, pero trabajamos con el supuesto que las fechas cercanas a la reutilización serán los últimos días del curso eliminado y los primeros del nuevo curso, los cuales no suelen ser de mucha actividad, y ante la duda, se encuadrarán en el nuevo curso.

De esta forma, la alternativa planteada consistirá en, tener una copia de la tabla de cursos de *Moodle*, en la que tan solo tendremos el identificador de los cursos activos (Tabla 76). Junto con este identificador guardaremos el identificador *BigData* con el que queremos guardar nuestros resultados. Al guardar estos resultados haremos un mapeo de identificador *Moodle* a

identificador *BigData* usando esta tabla y suponiendo que el dato que estamos analizando debe pertenecer al curso, con el identificador correspondiente, que esté activo y, por tanto, en nuestra tabla de cursos actuales. Quizá hubiera sido mejor práctica hacer este procesado de forma análoga a los usuarios, enumerado en el apartado 4.5.2, y guardar la información de *Moodle* directamente referida con el nuevo identificador, pero cuando se vio que esa alternativa era posible, el desarrollo del código estaba muy avanzado y suponía un enorme cambio. Puede ser este uno de los puntos de mejora del código. Además, la gestión de cursos actuales es un poco más tediosa que la de usuarios, ya que se debe gestionar la diferencia entre cursos individuales y cursos *META*.

moodle2_bigdata_results.cursos_actuales		
Campo	**Tipo**	**Descripción**
id	int	identificador que el curso tiene en *Moodle*
id_bigdata	string	identificador con el que referenciaremos a este curso en *BigData*

Tabla 76. Estructura de la tabla de cursos actuales (cursos_actuales)

Por otro lado, tendremos una tabla de historial de cursos que seguirá la estructura de la Tabla 77.

moodle2_bigdata_results.cursos_historico		
Campo	**Tipo**	**Descripción**
id_bigdata	string	identificador que la infraestructura *BigData* da al curso
nombre	string	nombre del curso
anio	string	año académico del curso, de la forma *Curso_20XX_20XX*
creación	date	momento en que el curso se creó o añadió a *Moodle*, obtenido a partir del campo timecreated de mdl_course. Formato YYYY-MM-dd
inicio	date	momento de inicio del curso, obtenido a partir del campo startdate de mdl_course. Formato YYYY-MM-dd

Tabla 77. Estructura de la tabla de historial de cursos (cursos_historico)

En cuanto a cómo gestionar estas tablas, en primer lugar, se analiza la tabla de cursos de *Moodle* para obtener identificadores activos y generar el identificador *BigData* que corresponde. Después, los identificadores de cursos *META*, se sustituyen por la concatenación de los identificadores de sus cursos hijos. Sabemos qué cursos son *META* a través de la tabla mdl_enrol (se puede ver la explicación completa en el código fuente del trabajo).

A partir de esta tabla se irán añadiendo entradas a la tabla de histórico de cursos. En concreto, se añadirán aquellos cursos que aparecen en la tabla de cursos actuales pero no en la de histórico, fijándonos en su identificador *BigData*. De esta forma, si un identificador *Moodle* se reasigna a un nuevo curso lo detectaremos porque el identificador *BigData* que se obtiene es distinto. Así, encontraremos en la tabla de cursos actuales un identificador *BigData* que no está en la tabla de histórico, por lo que accederemos a *Moodle* para obtener toda la información relativa al curso cuyo identificador mapeado a identificador *BigData* no está en nuestra tabla de histórico. Por simplicidad en el procesamiento, la información que se añade a la tabla de historial de cursos no cambia, es decir, no se ha contemplado la opción de que la parte de nombre de la asignatura sea modificado, por ejemplo, por un fallo de ortografía.

Quedan dos campos por analizar, el tiempo de creación y el tiempo de inicio, que se han incluido, simplemente, para mantener coherencia en los datos, es decir, para que cuando analicemos elementos de los cursos para los cuales necesitamos información sobre fechas de creación, estas sean coherentes con la fecha de creación de su curso, es decir, que no sean anterio-

res, y en caso de que lo sean (existen algunos procedimientos de creación a partir de elementos ya existentes que no actualizan la fecha de creación) sustituirlas por la fecha de creación del curso, como solución menos mala.

En caso de que el curso sea una reinicialización de contenido de otro curso existente, cabe la posibilidad de que no se actualice su fecha de creación, por ello, acudimos a la tabla de `UltimoDia` del análisis, donde tendremos el último día en que se actualizó la tabla de cursos, bajo la clave `cursos`. En caso de que la fecha de creación sea anterior a esta fecha sabremos que hay algo que no cuadra, porque hasta ese día, no tuvimos constancia de ese curso, por lo que podemos suponer que es un curso que se ha creado a partir de otro y no ha actualizado correctamente la fecha. Así, colocamos como fecha de creación del curso ese último día en que supimos que no estaba.

Consideremos un ejemplo, tenemos la tabla `mdl_course` de la Tabla 78 y la tabla `mdl_enrol` nos dice que el curso con identificador 2 es un curso *META* con cursos hijos 3 y 4 (por simplicidad no ha incluido su contenido completo aquí; para saber cómo nos da esta información, recurrir al código fuente). Con estos datos, obtendríamos la tabla `cursos_actuales` de la Tabla 79 y la tabla histórica de la Tabla 80.

mdl_course				
id	name	idnumber	timecreated	startdate
1	Curso 1 (0001-2015)	0001-2015	1443657600	1443657600
2	Curso 2 (META1-2015)	META1-2015	1441065600	1441065600
3	Curso 3 (0003-2015)	0003-2015	1441065600	1441065600
4	Curso 4 (0004-2015)	0004-2015	1441065600	1441065600

Tabla 78. Ejemplo de tabla de cursos `mdl_course` I

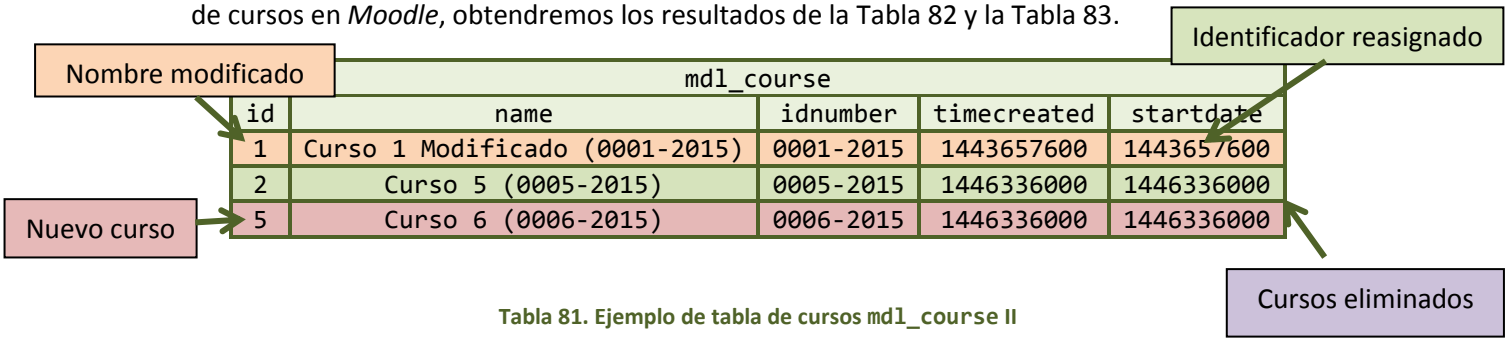
moodle2_bigdata_results.cursos_actuales	
id_bigdata	id
0001-2015	1
0003-2015//0004-2015	2
0003-2015	3
0004-2015	4

Tabla 79. Tabla `cursos_actuales` obtenida a partir de Tabla 78

moodle2_bigdata_results.cursos_historico				
id_bigdata	nombre	anio	creación	inicio
0001-2015	Curso 1 (0001-2015)	Curso_2015_2016	2015-10-01	2015-10-01
0003-2015//0004-2015	Curso 2 (META1-2015)	Curso_2015_2016	2015-09-01	2015-09-01
0003-2015	Curso 3 (0003-2015)	Curso_2015_2016	2015-09-01	2015-09-01
0004-2015	Curso 4 (0004-2015)	Curso_2015_2016	2015-09-01	2015-09-01

Tabla 80. Tabla `cursos_historico` obtenida a partir de Tabla 78

Ahora supongamos, que la Tabla 78 se modificada para dar lugar a la Tabla 81 como tabla de cursos en *Moodle*, obtendremos los resultados de la Tabla 82 y la Tabla 83.



mdl_course				
id	name	idnumber	timecreated	startdate
1	Curso 1 Modificado (0001-2015)	0001-2015	1443657600	1443657600
2	Curso 5 (0005-2015)	0005-2015	1446336000	1446336000
5	Curso 6 (0006-2015)	0006-2015	1446336000	1446336000

Tabla 81. Ejemplo de tabla de cursos `mdl_course` II

moodle2_bigdata_results.cursos_actuales	
id_bigdata	id
0001-2015	1
0005-2015	2
0006-2015	5

Tabla 82. Tabla cursos_actuales obtenida a partir de Tabla 81

moodle2_bigdata_results.cursos_historico				
id_bigdata	nombre	anio	creación	inicio
0001-2015	Curso 1 (0001-2015)	Curso_2015_2016	2015-10-01	2015-10-01
0003-2015//0004-2015	Curso 2 (META2-2015)	Curso_2015_2016	2015-09-01	2015-09-01
0003-2015	Curso 3 (0003-2015)	Curso_2015_2016	2015-09-01	2015-09-01
0004-2015	Curso 4 (0004-2015)	Curso_2015_2016	2015-09-01	2015-09-01
0005-2015	Curso 5 (0005-2015)	Curso_2015_2016	2015-11-01	2015-11-01
0006-2015	Curso 6 (0006-2015)	Curso_2015_2016	2015-11-01	2015-11-01

Tabla 83. Tabla cursos_historico obtenida a partir de Tabla 81

La **tabla de historial de matriculaciones** (bloque Matriculaciones, tabla moodle2_bigdata_results.matriculaciones) se encargará de mantener todas las asociaciones curso – usuario, es decir, indicará qué usuarios pueden acceder a qué curso, para saber qué estadísticas de accesos de qué usuario hay que recoger en cada curso. Presenta la estructura de la Tabla 84.

moodle2_bigdata_results.matriculaciones		
Campo	**Tipo**	**Descripción**
curso	string	identificador del curso
usuario	string	identificador de usuario
inicio	date	día en que se creó, en *Moodle*, la asociación. Realmente tomamos como referencia el instante de inicio de la asociación (timestart en mdl_user_enrolments). Formato YYYY-MM-dd
fin	date	día en que se eliminó la asociación. Mientras la asociación siga activa, este campo no tiene sentido. En mdl_user_enrolments existe un campo de fin del enrolamiento del usuario, pero normalmente no se usa, por lo que vamos a considerar que ese campo no existe, para no complicar el análisis. Formato YYYY-MM-dd
status	string	el usuario actúa como profesor ('PROFESOR') o alumno ('ALUMNO') del curso
activo	string	permite saber si para esta asociación es posible ('ACTIVO') o no ('NO ACTIVO') que se sigan produciendo registros

Tabla 84. Estructura moodle2_bigdata_results.matriculaciones

Juntando la información de las tablas mdl_enrol y mdl_user_enrolments podemos obtener qué usuarios pueden acceder a cada curso. La única diferencia es que en nuestra tabla guardaremos los cursos referidos con su identificador *BigData*, obtenido a partir de la tabla de cursos actuales. Obviamente, las asociaciones curso – usuario actuales que nos encontramos, se refieren a cursos actuales. Por otro lado, juntando la información de mdl_config, mdl_context y mdl_role_assignments se puede diferencian profesores de alumnos. Con la variable gradebookroles, definida en mdl_config, se define a qué roles (de los definidos en mdl_role) se les pueden asignar notas. Será esta la clave para diferenciar alumnos de profesores, ya que a los profesores no se les puede evaluar. Sabiendo la lista de roles "de tipo alumno" y qué rol toma cada usuario en cada curso, podremos saber que status tiene cada usuario en cada curso.

Esta tabla mantiene todas las asociaciones que en algún momento han existido en *Moodle*. Si una misma asociación ha existido durante dos rangos de tiempo distintos, encontraremos dos entradas para la misma pareja curso – usuario, pero con distintos rangos de actividad no solapados, y solamente una de ellas podrá tener `activo = 'ACTIVO'`. Cuando se borra la asociación (suponemos que cuando se borra el curso o el usuario implicados también se borra su asociación) se desactiva la asociación (`activo = 'NO ACTIVO'`) y se sustituye el día de fin de la asociación por el día anterior al actual. También se ha supuesto que el identificador de usuario es único y no se reasigna cuando desaparece un usuario.

Para generar este comportamiento, por un lado tendremos la inicialización de datos, la cual tienen por objetivo cargar todas las asociaciones encontradas en *Moodle* así como guardarlas sustituyendo el identificador del curso en *Moodle* por el identificador *BigData* del mismo.

Por ejemplo, supongamos que tenemos Tabla 85 y Tabla 86 como `mdl_user_enrolments` y `mdl_enrol`, respectivamente. Por otro lado, supongamos que a partir de las tablas `mdl_role_assignment` y `mdl_context` obtenemos que el usuario 1 actúa con rol 3 en los cursos 2 y 1, y los usuarios 2 y 3 actúan con rol 5 en cualquier curso; además la opción `gradebookroles` de `mdl_config`, dice que solo el rol 5 es evaluable (seguir el proceso de obtención en el código fuente, no incluido aquí por simplicidad y extensión), obtenemos los resultados de la Tabla 87.

mdl_user_enrolments		
enrolid	userid	timestart
3	1	1444471200
3	2	1444474800
4	1	1443697200
4	3	1443697200

Tabla 85. Ejemplo de `mdl_user_enrolments` I

mdl_enrol	
id	courseid
3	2
4	1

Tabla 86. Ejemplo de `mdl_enrol`

moodle2_bigdata_results.matriculaciones					
curso	usuario	inicio	fin	status	activo
0003-2015//0004-2015	1	2015-10-10	None	PROFESOR	ACTIVO
0003-2015//0004-2015	2	2015-10-10	None	ALUMNO	ACTIVO
0001-2015	1	2015-10-01	None	PROFESOR	ACTIVO
0001-2015	3	2015-10-01	None	ALUMNO	ACTIVO

Tabla 87. Tabla de matriculaciones obtenida a partir de Tabla 85 y Tabla 86

Asociación eliminada

mdl_user_enrolments		
enrolid	userid	timestart
3	2	1444474800
4	1	1443697200
4	3	1443697200
4	4	1446336000

Nueva asociación

Tabla 88. Ejemplo de `mdl_user_enrolments` II

En cuanto al análisis incremental, la parte central de ese análisis actualiza el contenido de la tabla uniendo los resultados de cuatro consultas. Supongamos que la Tabla 85 se ha modificado por la Tabla 88. En cuanto a la obtención del status, supongamos que para los usuarios anteriores da el mismo resultado, y cualquier nuevo usuario tiene rol 5.

El código implementado llevará a cabo la actualización en varias etapas.

- En primer lugar, se seleccionan aquellas matriculaciones que ya se han desactivado y no se deben modificar.

- A continuación, se cogen aquellas asociaciones cuya pareja curso – usuario sigue existiendo en *Moodle*. En nuestro ejemplo, se obtiene la Tabla 89.

curso	usuario	inicio	fin	status	activo
0003-2015//0004-2015	2	2015-10-10	None	ALUMNO	ACTIVO
0003-2015	1	2015-10-01	None	PROFESOR	ACTIVO
0003-2015	3	2015-10-01	None	ALUMNO	ACTIVO

Tabla 89. Resultado de la *sub – query* II sobre la Tabla 88

- En tercer lugar, se buscan aquellas matriculaciones que ya no existen para ser desactivadas y fijar su fecha de fin al día anterior al actual. En nuestro ejemplo se obtiene la Tabla 90.

curso	usuario	inicio	fin	status	activo
0003-2015//0004-2015	1	2015-10-10	2015-11-01	PROFESOR	NO ACTIVO

Tabla 90. Resultado de la *sub – query* III sobre la Tabla 88

- Por último, se añaden las nuevas asociaciones, incluidas aquellas que son idénticas a una ya existente pero ya desactivada. En este último caso, se comprueba su fecha de inicio y si esta es anterior a la fecha de fin de la misma asociación ya desactivada, se fija su fecha de inicio al día siguiente a la finalización de la anterior, para que no se solapen. En nuestro ejemplo se obtiene la Tabla 91.

curso	usuario	inicio	fin	status	activo
0001-2015	4	2015-11-01	None	ALUMNO	ACTIVO

Tabla 91. Resultado de la *sub – query* IV sobre la Tabla 88

Juntando todos los resultados, se actualizará el contenido de la tabla de matriculaciones al mostrado en la Tabla 92

moodle2_bigdata_results.matriculaciones					
curso	usuario	inicio	fin	status	activo
0003-2015//0004-2015	2	2015-10-10	None	ALUMNO	ACTIVO
0001-2015	1	2015-10-01	None	PROFESOR	ACTIVO
0001-2015	3	2015-10-01	None	ALUMNO	ACTIVO
0003-2015//0004-2015	1	2015-10-10	2015-11-01	PROFESOR	NO ACTIVO
0001-2015	4	2015-11-01	None	ALUMNO	ACTIVO

Tabla 92. Tabla de matriculaciones resultante a partir de Tabla 88 mediante análisis incremental

En cualquiera de los casos (inicial o incremental) si el instante de inicio de la asociación es anterior al inicio del curso, se considera inicio de la asociación el del curso. Es algo poco probable que ocurra, pero lo consideramos para mantener coherencia en los datos.

La **tabla de historial de cursos – módulos** (`Modulos`, tabla: `moodle2_bigdata_results.cursos_modulos`) mantiene información sobre todos los módulos que se han añadido en cada curso. Su estructura se muestra en la Tabla 93.

moodle2_bigdata_results.cursos_modulos		
Campo	**Tipo**	**Descripción**
curso	`string`	identificador de curso
modulo	`int`	identificador de módulo
identificador	`int`	identificador del contenido del módulo, según su categoría
tipo	`string`	tipo de contenido
nombre	`int`	nombre del contenido
inicio	`date`	momento en que se añadió el módulo al curso. Formato YYYY-MM-dd
fin	`date`	momento en que ya no se pueden producir accesos al módulo, pensado para cuando el curso ya no está activo, pero también puede ocurrir que directamente se borre el recurso. Formato YYYY-MM-dd
inicio_timestamp	`bigint`	marca de tiempo (fecha y hora) en que se añadió el módulo
activo	`string`	indica si es posible (`'ACTIVO'`) o no (`'NO ACTIVO'`) que se puedan seguir produciendo accesos al módulo

Tabla 93. Estructura `moodle2_bigdata_results.cursos_modulos`

En *Moodle* se pueden distinguir distintos tipos de módulos que contienen distintos formatos de información publicada para los usuarios. Guardamos la información de todos ellos junta. Para generar esta tabla debemos tener en cuenta la información de tres tablas:

- `mdl_course_modules`: mantiene información sobre los distintos módulos definidos en cada curso. Un módulo se puede considerar como un contenedor de algún tipo de elemento publicado en *Moodle*. Pueden contener elementos de tipo recurso, foro, *assign*, *quiz*, etc. El tipo de elemento viene dado mediante un identificador de tipo.

- `mdl_modules`: define la correspondencia entre identificador y tipo de módulo.

- Tablas de cada tipo de elemento. Cada tipo de elemento publicable en un módulo tiene su propia tabla donde se mantiene su información.

Para cada uno de los módulos publicados en cada curso guardaremos su identificador de módulo, el tipo de elemento que contiene y el identificador del tipo de elemento a partir de la información de `mdl_course_modules`. Sabiendo su tipo de elemento (a través de `mdl_modules`) accedemos a su nombre, para completar la información. Para ello, previamente se guardará la información de todos los nombres de todos los elementos publicados, referenciados mediante su tipo de elemento e identificador, en una única tabla.

Por otro lado, el campo de inicio (creación del módulo) se puede extraer de `mdl_course_module`. Se ha observado que cuando el módulo es copia de otro ya existente, la copia (con nuevos identificadores de módulo) guarda como fecha de creación, la de la copia original. Como para el cálculo de nuestras métricas, este valor es incoherente, se ha optado por suponer que, en caso de que la creación del módulo sea anterior a la creación del propio curso, la fecha de creación del curso y módulo se igualan, ya que no ha encontrado información para poder obtener una fecha más realista. Como ya se indicó, el curso se referencia por medio de su identificador *BigData*.

Por otro lado, cuando se detecta que el módulo ya no existe en *Moodle* (principalmente porque se ha borrado el curso, pero también puede ocurrir que se borre el módulo), la entrada correspondiente se desactiva (`activo = 'NO ACTIVO'`) y se sustituye su fecha de fin por el día anterior al actual. En la generación y actualización de esta tabla se ha supuesto que los identificadores de módulo no se reutilizan, al menos, dentro del mismo curso.

La inicialización de esta tabla se encargará de obtener la información previamente indicada directamente de *Moodle*. Por ejemplo, consideremos que hemos juntado todos los nombres de los módulos en la Tabla 94, tenemos la Tabla 95 como `mdl_course_modules` y la Tabla 96 como `mdl_modules`. A partir de ellas, obtenemos la Tabla 97.

moodle2_bigdata_results.nombre_modulos			
curso	id	nombre	tipo
2	1	Recurso 1 del curso 2	resource
2	1	Foro 1 del curso 2	forum
1	1	Recurso 1 del curso 1	resource
2	1	Assign 1 del curso 2	assign

Tabla 94. Ejemplo de `moodle2_bigdata_results.nombre_modulos`

mdl_course_modules				
id	course	module	instance	added
1	2	17	1	1444474800
2	2	9	1	1445338800
3	1	17	2	1444906800
4	2	1	1	1444906800

Tabla 95. Ejemplo de `mdl_course_module` I

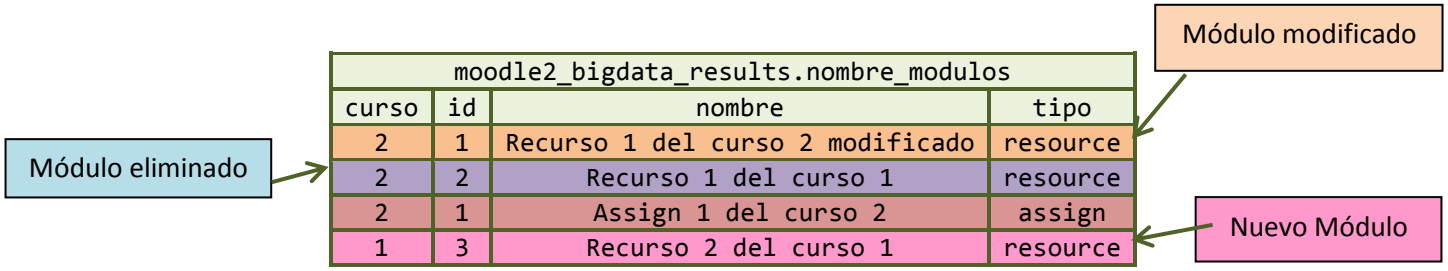
mdl_modules	
id	name
1	assign
9	forum
17	resource

Tabla 96. Ejemplo de `mdl_modules`

moodle2_bigdata_results.cursos_modulos								
curso	modulo	identi-ficador	tipo	nombre	inicio	fin	inicio_timestamp	activo
0003-2015//0004-2015	1	1	resource	Recurso 1 del curso 2	2015-10-10	None	1444474800	ACTIVO
0003-2015//0004-2015	2	1	forum	Foro 1 del curso 2	2015-10-20	None	1445338800	ACTIVO
0001-2015	3	2	resource	Recurso 1 del curso 1	2015-10-15	None	1444906800	ACTIVO
0003-2015//0004-2015	4	1	assign	Assign 1 del curso 2	2015-10-15	None	1444906800	ACTIVO

Tabla 97. Tabla de módulos obtenida a partir de Tabla 94, Tabla 95 y Tabla 96

Supongamos a partir de ahora, que la Tabla 94 y la Tabla 95 se sustituyen, respectivamente por la Tabla 98 y la Tabla 99.



moodle2_bigdata_results.nombre_modulos			
curso	id	nombre	tipo
2	1	Recurso 1 del curso 2 modificado	resource
2	2	Recurso 1 del curso 1	resource
2	1	Assign 1 del curso 2	assign
1	3	Recurso 2 del curso 1	resource

Tabla 98. Ejemplo de mdl_resource II

mdl_course_modules				
id	course	module	instance	added
1	2	17	1	1444474800
3	1	17	2	1444906800
4	2	1	1	1444906800
5	1	17	3	1446336000

Tabla 99. Ejemplo de mdl_course_modules II

Una vez inicializada la tabla, los análisis incrementales realizarán el siguiente trabajo:

- Obtención de aquellas entradas que ya fueron desactivadas y que ya no se van a modificar.
- Obtención de las relaciones curso – módulo que siguen existiendo actualizando el nombre del módulo por si se hubiera modificado en *Moodle*. Para nuestro ejemplo obtenemos la Tabla 100.

curso	modulo	identificador	tipo	nombre	inicio	fin	inicio_timestamp	activo
0003-2015//0004-2015	1	1	resource	Recurso 1 del curso 2 Modificado	2015-10-10	None	1444474800	ACTIVO
0001-2015	3	2	resource	Recurso 1 del curso 1	2015-10-15	None	1444906800	ACTIVO
0003-2015//0004-2015	4	1	assign	Assign 1 del curso 2	2015-10-15	None	1444906800	ACTIVO

Tabla 100. Resultado de la *sub – query* II a partir de Tabla 98 y Tabla 99

- Búsqueda de los módulos que ya no existen en *Moodle* para ser desactivados y fijar su fecha de fin.

curso	modulo	identificador	tipo	nombre	inicio	fin	inicio_timestamp	activo
0003-2015//0004-2015	2	1	forum	Foro 1 del curso 2	2015-10-20	2015-11-01	1445338800	NO ACTIVO

Tabla 101. Resultado de la *sub – query* III a partir de Tabla 98 y Tabla 99

- Adición de las relaciones curso – módulo todavía no existentes en la tabla, es decir, aquellos módulos que se han añadido a *Moodle* desde el último de nuestros análisis.

curso	modulo	identi-ficador	tipo	nombre	inicio	fin	inicio_timestamp	activo
0001-2015	5	3	resource	Recurso 2 del curso 3	2015-11-01	None	1446336000	ACTIVO

Tabla 102. Resultado de la *sub – query* IV a partir de Tabla 98 y Tabla 99

El resultado final se muestra en la Tabla 103.

moodle2_bigdata_results.cursos_modulos								
curso	modulo	identi-ficador	tipo	nombre	inicio	fin	inicio_timestamp	activo
0003-2015//0004-2015	1	1	resource	Recurso 1 del curso 2 Modificado	2015-10-10	None	1444474800	ACTIVO
0001-2015	3	2	resource	Recurso 1 del curso 1	2015-10-15	None	1444906800	ACTIVO
0003-2015//0004-2015	4	1	assign	Assign 1 del curso 2	2015-10-15	None	1444906800	ACTIVO
0003-2015//0004-2015	2	1	forum	Foro 1 del curso 2	2015-10-20	2015-11-01	1445338800	NO ACTIVO
0001-2015	5	3	resource	Recurso 2 del curso 3	2015-11-01	None	1446336000	ACTIVO

Tabla 103. Tabla de módulos resultante a partir de Tabla 98 y Tabla 99 mediante análisis incremental

La **tabla de historial de cursos – foros** (Foros, tabla: `moodle2_bigdata_results.cursos_foros`) mantiene información sobre todos los foros que se han añadido en cada curso. Su estructura se muestra en la Tabla 104.

moodle2_bigdata_results.cursos_foros		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
foro	int	identificador del módulo que contiene al foro
identificador	int	identificador del foro
nombre	int	nombre del foro
inicio	date	momento en que se añadió el foro al curso. Formato YYYY-MM-dd
fin	date	momento en que ya no se pueden producir accesos al foro, pensado para cuando el curso ya no está activo, pero también puede ocurrir que directamente se borre el foro. Formato YYYY-MM-dd
inicio_timestamp	bigint	marca de tiempo (fecha y hora) en que se añadió el foro
activo	string	indica si es posible (‘ACTIVO’) o no (‘NO ACTIVO’) que se puedan seguir produciendo accesos al foro

Tabla 104. Estructura `moodle2_bigdata_results.cursos_foros`

A pesar de que en la tabla de módulos ya mantenemos la información de los foros, como estos son un caso especial de módulos sobre lo que se puede producir una actividad más distinta que en el resto de módulos, los vamos a analizar por separado, por lo que generamos esta tabla, para poder mantenerlos a parte del resto de módulos. Obviamente, esta tabla se calcula a partir de la tabla de módulos, seleccionando solo los módulos de tipo foro (*forum*).

La **tabla de historial de cursos – discusiones** (`Discusiones`, tabla: `moodle2_bigdata_results.cursos_foros_discusiones`) mantiene información sobre todas las discusiones que se han añadido en cada curso. Su estructura se muestra en la Tabla 93.

moodle2_bigdata_results.cursos_foros_discusiones		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
foro	int	identificador del módulo que contiene al foro al que pertenece la discusión
identificador	int	identificador del foro al que pertenece la discusión
discusion	int	identificador de la discusión
nombre	int	nombre de la discusión
inicio	date	momento en que se añadió la discusión al curso. Formato YYYY-MM-dd
usuario	string	identificador del usuario que crea la discusión
fin	date	momento en que ya no se pueden producir accesos la discusión, pensado para cuando el curso ya no está activo, pero también puede ocurrir que directamente se borre el foro o la discusión. Formato YYYY-MM-dd
activo	string	indica si es posible (‘ACTIVO’) o no (‘NO ACTIVO’) que se puedan seguir produciendo accesos a la discusión

Tabla 105. Estructura `moodle2_bigdata_results.cursos_foros_discusiones`

Los foros se componen de discusiones o hilos donde se generan conversaciones entre los usuarios. Estas discusiones no son tratadas como módulos, así que las registramos en una tabla propia, cuya generación y actualización es análoga a la de módulos (a partir de `mdl_forum`, `mdl_forum_discussions` y `mdl_forum_post`), detallada anteriormente y por ello no se entra en más detalle. Para mayor información, recurrir al código fuente del trabajo.

La tabla de **historial de *Assign*** (`Assign`, tabla `moodle2_bigdata_results.cursos_assign`) mantiene información sobre las tareas (*assign*) definidas en cada curso para que los usuarios envíen sus trabajos. Solo se mantienen los *assign* que están actualmente en el curso, es decir, si un profesor borra un *assign*, también se borrará de nuestro historial, ya que los *assign* de esta tabla se consideran a nivel de resultados académicos, y si un profesor borrar el *assign* se supone que es que considera que no es relevante para la nota final del alumno, por lo que se borra de nuestro análisis. También mantiene la lista de *assign* de los cursos ya no presentes en *Moodle*, manteniendo la última lista que se vio para ellos en *Moodle*. Su estructura se muestra en la Tabla 106.

moodle2_bigdata_results.cursos_assign		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
assign	int	identificador de *assign*
name	string	nombre del *assign*
duedate	bigint	fecha máxima de entrega del *assign*, en formato *timestamp*
teamsubmision	tinyint	permite saber si la entrega al *assign* es individual (0) o en grupo (1)

Tabla 106. Estructura `moodle2_bigdata_results.cursos_assign`

Los *assign* se guardan identificados con su identificador como *assign* y no mediante el identificador del módulo que los contiene, ya que en el proceso donde se va utilizar esta información, los *assign* vienen referenciados por su identificador. Mientras en la tabla de módulos se

mantienen todos los *assign* que han existido, aunque ya se hayan eliminado, en esta guardamos solo los *assign* actuales, junto con información concreta de ellos, que no podemos almacenar en la tabla de módulos, ya que no es un parámetro común al resto de módulos. De esta forma, a partir de la tabla de módulos analizaremos las visitas a los *assign*, pero lo realmente importante de ellos son las entregas de usuarios y las calificaciones obtenidas por ellos, algo que se realiza tomando como referencia esta tabla. Su obtención es directa de la tabla `mdl_assign`, por lo que no se presenta ningún ejemplo gráfico. En cuanto a los análisis incrementales, estos simplemente separarán el proceso en dos partes:

- Obtener, de la versión anterior de la tabla, los *assign* de cursos ya no activos, para mantener su información.
- Obtener de *Moodle* la información para los cursos actuales y añadirla a la anterior.

La tabla de **historial de *Quiz*** (`Quiz`, tabla: `moodle2_bigdata_results.cursos_quiz`) guarda información sobre los *quiz* publicados en cada curso para ser resueltos por los usuarios. De igual forma que ocurría con los *assign*, solo se guarda información sobre los *quiz* actuales, por lo que si un *quiz* se borra de *Moodle*, se dejará de tener en cuenta. Su estructura se presenta en la Tabla 107.

moodle2_bigdata_results.cursos_quiz		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
quiz	int	identificador de *quiz*
name	string	nombre del *quiz*

Tabla 107. Estructura moodle2_bigdata_results.cursos_quiz

Todas las cuestiones relativas a la tabla de *Assign* se pueden aplicar a la tabla de *Quiz*, obteniendo la información de `mdl_quiz`.

La tabla de **grupos** (`Grupos`, tabla: `moodle2_bigdata_results.grupos`) realmente no es un historial ya que solo mantiene información que actualmente se encuentra en *Moodle* pero reordenada. En *Moodle* la información de grupos se reparte por dos tablas:

- `mdl_groups`, donde se definen los grupos de trabajo por curso.
- `mdl_groups_members`, donde se define qué usuario pertenece a cada grupo.

En esta tabla de grupos se mantiene esta información junta, siguiendo la estructura de la Tabla 108.

moodle2_bigdata_results.grupos		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
grupo	int	identificador de grupo de trabajo
usuario	string	identificador de un usuario que pertenece al grupo

Tabla 108. Estructura moodle2_bigdata_results.grupos

Solo se mantiene la información actual, ya que se necesita para realizar una serie de cálculos, para los cuáles se necesita la información actual de *Moodle*. Por no complicar la generación de la tabla, ya que los cálculos a los que contribuye no son de vital importancia que sean precisos, se ha optado por dejarlo así.

4.6.3. Análisis de las estadísticas de acceso

Una vez sentadas las bases de nuestra estrategia de trabajo así como las herramientas que vamos a utilizar, en primer lugar procedemos a realizar un análisis estadístico de los accesos de los usuarios (tanto alumnos como profesores, sin distinción entre ambos), haciendo uso de las tablas descritas en los apartados anteriores y `mdl_log`.

En concreto, generamos estadísticas bajo distintas condiciones:

- Estadísticas por día – curso – usuario – módulo.
- Estadísticas por día – curso – usuario – foro – discusión.
- Estadísticas por día – curso – usuario – foro.
- Estadísticas por día – curso – usuario.
- Estadísticas por curso – usuario – módulo.
- Estadísticas por curso – módulo.
- Estadísticas por curso – usuario – foro – discusión.
- Estadísticas por curso – usuario – foro.
- Estadísticas por curso – usuario.
- Estadísticas por curso.

Las cuatro primeras estadísticas de esta lista constituyen el registro simulado de *Moodle* que se ha comentado al principio de esta apartado. Es decir, es una versión más ordenada de la información del registro de *Moodle*, `mdl_log`. De toda la información de accesos que nos proporciona este registro, nos quedamos con aquellos, que a nuestro criterio, son más interesantes. Estas cuatro tablas de estadísticas acceden a `mdl_log`, recogen la información de interés y la guardan ordenadamente. El resto de estadísticas de acceso se calculan a partir de ellas, por lo que no dependen `mdl_log`. De esta forma, si queremos calcular el total de accesos realizados por un usuario a un curso, no hace falta que recurramos a `mdl_log`, podemos, por ejemplo, recurrir a las estadísticas por día – curso – usuario, donde tenemos asegurados tener información desde el comienzo de funcionamiento de la aplicación, mientras que de `mdl_log` es posible que se hayan borrado los registros más antiguos.

Dadas estas estadísticas detalladas por día, el resto de métricas se recalculan a partir de ellas, cada día. En este punto se plantearon dos alternativas: cada día se quiere, por ejemplo, calcular el total de accesos de un usuario a un curso, y suponiendo que ya tenemos el total de accesos hasta el día anterior, podemos:

- Calcular el total de accesos para el día de hoy y sumarlos a los de ayer.
- Realizar el cálculo desde el primer día que estamos registrando.

Se ha optado por la segunda opción, ya que la primera implica que el cálculo hasta el día anterior fuera correcto. Si no lo es, por cualquier error de ejecución de la aplicación, se irá acumulando error análisis a análisis. El cálculo desde cero, se plantearía como más costoso, porque tiene que recorrer toda la información para realizar la suma, a pesar que la mayor parte de esa suma ya está hecha, pero nos asegura que el cálculo día a día será correcto. Un día

puede ser incorrecto, por ejemplo, porque las estadísticas originarias estaban incompletas, pero este problema se soluciona al día siguiente, por lo que se vuelve a los valores correctos.

Además, para todas estas métricas se ha construido una tabla particionada por día, donde cada día que se calculen las métricas, no se sustituya el cálculo del análisis anterior, sino que se añada junto con la fecha del análisis, para poder ver la evolución de las estadísticas.

Para la generación de estas tablas, se ha necesitado la intervención de una serie de tablas o bloques intermedios, con el objetivo de generar resultados parciales sencillos y no implementar sentencias *HiveQL* muy complicadas. Este apartado de estadísticas ha sido realizado con bastante detalle, quizá demasiado para lo que después vamos a necesitar para obtener otros resultados, pero se ha optado por realizarlo así, para hacerlo más ordenado y dejar la puerta abierta a otros análisis. Por ejemplo, se ha detallado mucho el uso de los foros, pero la mayor parte de resultados no se utilizan para ningún procesado avanzado, pero así en un futuro va a ser más fácil añadir cualquier análisis sobre cómo los alumnos usan los foros, porque las bases del análisis ya están preparadas para ello.

En cuanto a la obtención de las métricas aquí tratadas, no se presentará ningún ejemplo ilustrativo, para evitar prolongar demasiado la extensión de este documento, ya que se consideran bastante fáciles de entender a partir del código fuente donde se puede encontrar convenientemente comentadas todas las decisiones de programación tomadas.

Estadísticas por día – curso – usuario – módulo (`EstDCUM`, tabla: `moodle2_bigdata_results.moodle_estadisticas_acceso_dia_curso_usuario_modulo`). Mantiene todas las estadísticas de acceso a los módulos detalladas por día, curso, usuario y módulo. Es decir, para cada usuario en cada curso, analiza el número de accesos que realizó cada uno de los días. Se guardan solo los accesos de tipo “vista”, es decir los que suponen una lectura del contenido del módulo, por ejemplo:

- para los módulos de tipo recurso no se tienen en cuenta las acciones de tipo actualización, borrado, etc.;
- para los módulos de tipo *assign* no se tiene en cuenta la entrega de la tarea;
- para los módulos de tipo foro no se tiene en cuenta la creación de un *post* o una discusión.

Se consideran solo los accesos de tipo “vista” ya que son el único tipo de acceso común a todos los módulos y el que se puede considerar de más interés dado que se va a utilizar para ver la influencia de un módulo sobre los resultados académicos. De un módulo que represente un libro, no interesa el número de veces que un usuario ha modificado el archivo correspondiente, solo el número de veces que lo ha leído, además, en este caso, es de suponer que sólo los profesores podrán hacer operaciones de modificación. Con criterios similares, para los foros solo se almacena el número de visitas al foro, el resto de acciones de interés (creación de mensajes, etc.) se manejan por separado ya que no son eventos comunes al resto de módulos. Principalemente, el escoger solo las acciones de tipo “vista” tiene por objetivo no incluir acciones que no estén relacionadas con la lectura del contenido del módulo, para que el cálculo de la influencia de este elementos sobre cualquiera nota, sea un poco más real. Por ejemplo:

- En elementos evaluables como assign o quiz, es de más interés el número de veces que el usuario ha leído el contenido del ejercicio que considerarlo junto con el número de veces que ha modificado o entregado la entrega.

- Recursos, libro, apuntes o similares. son elementos que se pueden modificar, eliminar, actualizar, crear, etc., pero de vistas a cómo influyen sobre la nota final, tiene sentido tomar solo las acciones de lectura.

Se construye teniendo como base la PlantillaDCUM, y calculando el número de accesos a partir de mdl_log. Presenta la estructura de la Tabla 109.

moodle2_bigdata_results. moodle_estadisticas_acceso_dia_curso_usuario_modulo		
Campo	**Tipo**	**Descripción**
dia	date	Formato YYYY-MM-dd
curso	string	identificador de curso
usuario	string	identificador de usuario
modulo	int	identificador de módulo
num_accesos_modulo	int	número de accesos al módulo realizados por el usuario

Tabla 109. Estructura moodle2_bigdata_results.moodle_estadisticas_acceso_dia_curso_usuario_modulo

Estadísticas por día – curso – usuario – foro – discusión (EstDCUFD, tabla: moodle2_bigdata_results.moodle_estadisticas_acceso_dia_curso_usuario_foro_discusion). Mantiene estadísticas sobre el uso de las discusiones, detallado por día, curso y usuario. Es decir, para cada discusión indica la actividad realizada por cada usuario, en cada curso y por día. Se calcula tomando datos de mdl_log y la PlantillaDCUFD. Sigue la estructura de la Tabla 110.

moodle2_bigdata_results. moodle_estadisticas_acceso_dia_curso_usuario_foro_discusion		
Campo	**Tipo**	**Descripción**
dia	date	Formato YYYY-MM-dd
curso	string	identificador de curso
usuario	string	identificador de usuario
foro	int	identificador del módulo que contiene al foro en que está definida la discusión
discusión	int	identificador de la discusión
num_accesos_discusion	int	número de accesos realizados por el usuario a la discusión
num_post_creados	int	número de *post* (mensajes) creados por el usuario en la discusión

Tabla 110. Estructura moodle2_bigdata_results.moodle_estadisticas_acceso_dia_curso_usuario_foro_discusion

Estadísticas por día – curso – usuario – foro (EstDCUF, tabla: moodle2_bigdata_results.moodle_estadisticas_acceso_dia_curso_usuario_foro). Mantiene las estadísticas relativas a la actividad llevada a cabo sobre cada foro, por cada usuario en cada curso, detallado por día. Se obtiene a partir de EstDCUFD, mdl_log y usando la PlantillaDCUF. Su estructura se presenta en la Tabla 111.

moodle2_bigdata_results. moodle_estadisticas_acceso_dia_curso_usuario_foro		
Campo	**Tipo**	**Descripción**
dia	date	Formato YYYY-MM-dd
curso	string	identificador de curso
usuario	string	identificador de usuario
foro	int	identificador del módulo que contiene al foro
num_accesos_foro	int	número de accesos realizados por el usuario a la página principal del foro, donde se listan las discusiones que contiene
num_discusiones_creadas	int	número de discusiones creadas por el usuario en el foro
num_accesos_discusiones	int	número de accesos realizados por el usuario a todas las discusiones definidas en el foro
num_post_creados	int	número de *post* creados por el usuario en el total de discusiones presentes en el foro

Tabla 111. Estructura moodle2_bigdata_results.moodle_estadisticas_acceso_dia_curso_usuario_foro

Las métricas num_accesos_foro y num_discusiones_creadas se obtienen de mdl_log, y num_accesos_discusiones y num_post_creados se obtienen de EstDCUFD.

Con respecto a la diferencia entre las EstDCUM y las EstDCUF, las primeras solo mantienen información sobre los accesos de lectura a los módulos, entre los cuales se guardará información de los accesos de lectura a los foros, es decir, cualquier acción de lectura en el foro, por ejemplo, se guarda conjuntamente las visitas a la página principal del foro y a cualquiera de sus discusiones. Las EstDCUF tratan los foros como un módulo con un comportamiento ligeramente distinto a los demás, ya que no solo es importante tener en cuenta las acciones de lectura sino también las de creación. Además de añadir estas acciones, se desglosan los distintos tipos de lecturas, ya que no es lo mismo leer la página principal de un foro, donde se listan sus discusiones, que acceder a una discusión y leer el contenido de sus mensajes.

Estadísticas por día – curso – usuario (EstDCU, tabla: moodle2_bigdata_results.moodle_estadisticas_acceso_dia_curso_usuario). Mantiene información sobre la actividad de cada usuario en cada curso. Obtiene información de diversos orígenes, detallado a continuación. Para generar la información toma como base la PlantillaDCU. Sigue la estructura de la Tabla 112.

moodle2_bigdata_results. moodle_estadisticas_acceso_dia_curso_usuario		
Campo	**Tipo**	**Descripción**
dia	date	Formato YYYY-MM-dd
curso	string	identificador de curso
usuario	string	identificador de usuario
num_accesos_totales	int	número de accesos totales, de cualquier tipo, que el usuario ha realizado al curso, en este día
num_accesos_únicos	int	número de accesos únicos realizados por el usuario al curso, en este día. El acceso único se mide al día, es decir, si un usuario realiza varias veces la misma acción el mismo día, solo cuenta una vez. Es una medida del número de cosas distintas que el usuario ha hecho al día

num_accesos_course_view	int	número de accesos realizados por el usuario a la página principal del curso, al día
num_accesos_modulos	int	número de accesos realizados por el usuario a los módulos con el objetivo de leer su contenido
num_accesos_fuera	int	número de accesos realizados por el usuario desde fuera de la universidad, es decir, aquellos cuya IP de origen no está dentro del conjunto de IPs proporcionadas a la Universidad de Valladolid
num_accesos_dentro	int	número de accesos realizados por el usuario desde dentro de la universidad, es decir, aquellos cuya IP de origen está dentro del conjunto de IPs proporcionadas a la Universidad de Valladolid
porc_accesos_fuera	double	porcentaje de accesos realizados por el usuario desde fuera de la universidad con respecto al total, expresado en tanto por uno
porc_accesos_dentro	double	porcentaje de accesos realizados por el usuario desde dentro de la universidad con respecto al total, expresado en tanto por uno
num_accesos_foros	int	número de visitas realizadas por el usuario a la página principal de los foros publicados en el curso
num_discusiones_creadas	int	número de discusiones creadas por el usuario en cualquiera de los foros del curso
num_accesos_discusiones	int	número de visitas realizadas por el usuario a cualquiera de las discusiones publicadas en el curso
num_post_creados	int	número de *post* creados por el usuario en cualquiera de las discusiones del curso
num_accesos_apuntes	int	número de accesos realizados por el usuario al total de módulos que contiene información que se puede considerar apuntes (en el código fuente se detalla la elección de estos módulos)

Tabla 112. Estructura `moodle2_bigdata_results.moodle_estadisticas_acceso_dia_curso_usuario`

En cuanto a la obtención de estas métricas:

- `num_accesos_totales`, `num_accesos_unicos`, `num_accesos_course_view`, `num_accesos_fuera` y `num_accesos_dentro` se obtienen de `mdl_log`.
- `num_accesos_modulos` y `num_accesos_apuntes` se obtienen de `EstDCUM`.
- `num_accesos_foros`, `num_discusiones_creadas`, `num_accesos_discusiones` y `num_post_creados` se obtienen a partir de `EstDCUF`

Estadísticas por curso – usuario – módulo (`EstCUM`, tabla: `moodle2_bigdata_results.moodle_estadisticas_acceso_curso_usuario_modulo`). Mantiene información sobre el uso que cada usuario ha hecho de cada módulo publicado en el curso, solo teniendo en cuenta los módulos a los que puede acceder en función de que coincidan temporalmente. Su estructura se presenta en la Tabla 113.

En cuanto a la obtención de estas métricas:

- `num_acceso_modulos` se obtiene de `EstDCUM`.

- fecha_primera_visita, fecha_media_acceso, diferencia_fecha_acceso, publicacion_modulo, diferencia_publicacion se obtienen de un bloque intermedio denominado PrimeraVisita que tiene su origen en mdl_log y CursoModuloUsuario.

moodle2_bigdata_results. moodle_estadisticas_acceso_curso_usuario_modulo		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
usuario	int	identificador de usuario
modulo	int	identificador de módulo
num_accesos_modulo	int	número de accesos realizados por el usuario al módulo
fecha_primera_visita	string	fecha en que el usuario accedió por primera vez al módulo, en formato YYYY-MM-dd HH:mm:ss. Si no accedió: "Nunca"
fecha_media_acceso	string	fecha media de acceso de los usuarios del curso al módulo, en formato YYYY-MM-dd HH:mm:ss. Si ninguno accedió o no se obtuvo información, "Sin datos"
diferencia_fecha_acceso	string	diferencia, en segundos, entre los dos campos anteriores. La resta se realiza en el sentido fecha_media_acceso – fecha_primera_visita. Si el usuario no accedió, "No aplicable"
publicacion_modulo	string	fecha de publicación del módulo, en formato YYYY-MM-dd HH:mm:ss.
diferencia_publicación	string	diferencia, en segundos, en el acceso al módulo con respecto a su publicación. La resta se realiza en el sentido fecha_primera_visita – publicacion_modulo. Si el usuario no accedió, "No aplicable"

Tabla 113. Estructura moodle2_bigdata_results.moodle_estadisticas_acceso_curso_usuario_modulo

Estadísticas por curso – módulo (EstCM, tabla: moodle2_bigdata_results.moodle_estadisticas_acceso_curso_modulo). Guarda información sobre la actividad que el total de usuarios ha hecho sobre cada módulo de cada curso. Presenta la estructura de la Tabla 114.

En cuanto a la obtención de estas métricas.

- num_accesos_modulo se obtiene de EstCUM.
- num_usuarios_total, num_usuarios_lecturas y porc_usuarios_lecturas se calculan por medio de un bloque intermedio denominado PorcAccesoModulos, que también tiene su origen en EstCUM.

moodle2_bigdata_results. moodle_estadisticas_acceso_curso_modulo		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
modulo	int	identificador de módulo
num_accesos_modulo	int	número de accesos al módulo
num_usuarios_total	int	número total de usuarios que pueden acceder al módulo
num_usuarios_lecturas	int	número de usuarios que han accedido, al menos una vez, al módulo
porc_usuarios_lecturas	double	porcentaje de usuarios que han accedido, al menos una vez, al módulo

Tabla 114. Estructura moodle2_bigdata_results.moodle_estadisticas_acceso_curso_modulo

Estadísticas por curso – usuario – foro – discusión (EstCUFD, tabla: moodle2_bigdata_results.moodle_estadisticas_acceso_curso_usuario_foro_discusion). Almacena las estadísticas de acceso de los usuarios a las discusiones publicadas en cada curso. Presenta la estructura de la Tabla 115. Se calcula exclusivamente a partir de EstDCUFD.

moodle2_bigdata_results. moodle_estadisticas_acceso_curso_usuario_foro_discusion		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
usuario	string	identificador de usuario
foro	int	identificador del módulo que contiene al foro en que está definida la discusión
discusión	int	identificador de la discusión
num_accesos_discusion	int	número de accesos realizados por el usuario a la discusión
num_post_creados	int	número de *post* (mensajes) creados por el usuario en la discusión

Tabla 115. Estructura moodle2_bigdata_results.moodle_estadisticas_acceso_curso_usuario_foro_discusion

Estadísticas por curso – usuario – foro (EstCUF, tabla: moodle2_bigdata_results.moodle_estadísticas_acceso_curso_usuario_foro). Mantiene las estadísticas de acceso a los foros, detalladas por curso y usuario. Presenta la estructura de la Tabla 116.

moodle2_bigdata_results.moodle_estadisticas_acceso_curso_usuario_foro		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
usuario	string	identificador de usuario
foro	int	identificador del módulo que contiene al foro
num_accesos_foro	int	número de accesos realizados por el usuario a la página principal del foro, donde se listan las discusiones que contiene
num_discusiones_creadas	int	número de discusiones creadas en el foro, por el usuario
num_accesos_discusiones	int	número de accesos del usuario a todas las discu-

		siones definidas en el foro
num_post_creados	int	número de *post* creados por el usuario en el total de discusiones presentes en el foro
num_discusiones_total	int	número total de discusiones definidas en el foro, a las que puede acceder el usuario
num_discusiones_leidas	int	número de discusiones que han sido leídas, al menos una vez, por el usuario
porc_discusiones_leidas	int	porcentaje de discusiones que han sido leídas, al menos una vez, por el usuario, en tanto por uno

Tabla 116. Estructura moodle2_bigdata_results.moodle_estadisticas_acceso_curso_usuario_foro

En cuanto a la obtención de estas métricas:

- num_accesos_foro, num_discusiones_creadas, num_accesos_discusiones y num_post_creados se obtienen a partir de EstDCUF.
- num_discusiones_total, num_discusiones_leidas y porc_discusiones_leidas se obtienen a partir de un bloque denominado PorcDiscusionesAccedidasCUF que tiene su origen en EstCUFD y CursoUsuarioForo.

Estadísticas por curso – usuario (EstCU, tabla: moodle2_bigdata_results.moodle_estadisticas_acceso_curso_usuario). Guarda la información de acceso detallada por curso y usuario. Presenta la estructura de la Tabla 117.

moodle2_bigdata_results.moodle_estadisticas_acceso_curso_usuario		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
usuario	string	identificador de usuario
num_accesos_totales	int	número de accesos totales, de cualquier tipo que el usuario ha realizado al curso
num_accesos_únicos	int	número de accesos únicos realizados por el usuario al curso. El acceso único se mide al día, es decir, si un usuario realiza varias veces la misma acción el mismo día, solo cuenta una vez. Es una medida del número de cosas distintas que el usuario ha hecho al día
num_accesos_course_view	int	número de accesos realizados por el usuario al página principal del curso
num_accesos_modulos	int	número de accesos realizados por el usuario a los módulos con el objetivo de leer su contenido
num_accesos_fuera	int	número de accesos realizados por el usuario desde fuera de la universidad, es decir, aquellos cuya IP de origen no está dentro del conjunto de IPs proporcionadas a la Universidad de Valladolid
num_accesos_dentro	int	número de accesos realizados por el usuario desde dentro de la universidad, es decir, aquellos cuya IP de origen está dentro del conjunto de IPs proporcionadas a la Universidad de Valladolid

porc_accesos_fuera	double	porcentaje de accesos realizados por el usuario desde fuera de la universidad con respecto al total, expresado en tanto por uno
porc_accesos_dentro	double	porcentaje de accesos realizados por el usuario desde dentro de la universidad con respecto al total, expresado en tanto por uno
num_accesos_foros	int	número de visitas realizadas por el usuario a la página principal de los foros publicados en el curso
num_discusiones_creadas	int	número de discusiones creadas por el usuario en cualquiera de los foros del curso
num_accesos_discusiones	int	número de visitas realizadas por el usuario a cualquiera de las discusiones publicadas en el curso
num_post_creados	int	número de post creados por el usuario en cualquiera de las discusiones del curso
num_accesos_apuntes	int	número de accesos realizados por el usuario al total de módulos que contiene información que se puede considerar apuntes (en el código fuente se detalla la elección de estos módulos)
num_accesos_lunes	int	número de accesos realizados por el usuario en días que fueron Lunes
porc_accesos_lunes	double	porcentaje de accesos realizados por el usuario en días que fueron Lunes, en tanto por uno
num_accesos_martes	int	número de accesos realizados por el usuario en días que fueron Martes
porc_accesos_martes	double	porcentaje de accesos realizados por el usuario en días que fueron Martes, en tanto por uno
num_accesos_miercoles	int	número de accesos realizados por el usuario en días que fueron Miércoles
porc_accesos_miercoles	double	porcentaje de accesos realizados por el usuario en días que fueron Miércoles, en tanto por uno
num_accesos_jueves	int	número de accesos realizados por el usuario en días que fueron Jueves
porc_accesos_jueves	double	porcentaje de accesos realizados por el usuario en días que fueron Jueves, en tanto por uno
num_accesos_viernes	int	número de accesos realizados por el usuario en días que fueron Viernes
porc_accesos_viernes	double	porcentaje de accesos realizados por el usuario en días que fueron Viernes, en tanto por uno
num_accesos_sabado	int	número de accesos realizados por el usuario en días que fueron Sábado
porc_accesos_sabado	double	porcentaje de accesos realizados por el usuario en días que fueron Sábado, en tanto por uno
num_accesos_domingo	int	número de accesos realizados por el usuario

		en días que fueron Domingo
porc_accesos_domingo	double	porcentaje de accesos realizados por el usuario en días que fueron Domingo, en tanto por uno
num_modulos_total	int	número total de módulos a los que puede acceder el usuario
num_modulos_leidos	int	número total de módulos accedidos o leídos, al menos una vez, por el usuario
porc_modulos_leidos	double	porcentaje de módulos accedidos o leídos, al menos una vez, por el usuario, en tanto por uno
num_foros_total	int	número total de foros a los que puede acceder el usuario
num_foros_accedidos	int	número total de foros accedidos, al menos una vez, por el usuario
porc_foros_accedidos	double	porcentaje de foros accedidos, al menos una vez, por el usuario, en tanto por uno
num_discusiones_total	int	número total de discusiones a las que puede acceder el usuario
num_discusiones_accedidas	int	número total de discusiones accedidas, al menos una vez, por el usuario
porc_discusiones_accedidas	double	porcentaje de discusiones accedidas, al menos una vez, por el usuario, en tanto por uno
num_apuntes_total	int	número total de apuntes a los que puede acceder el usuario
num_apuntes_leidos	int	número total de apuntes leídos, al menos una vez, por el usuario
porc_apuntes_leidos	double	porcentaje de apuntes leídos, al menos una vez, por el usuario, en tanto por uno.

Tabla 117. Estructura `moodle2_bigdata_results.moodle_estadisticas_acceso_curso_usuario`

En cuanto a la obtención de estas métricas:

- De `num_accesos_totales` a `num_accesos_apuntes` se obtienen a partir de `EstDCU`.
- Los accesos según día de la semana se obtienen de un bloque intermedio denominado `DiaSemana` que tiene su origen en `EstDCU`.
- `num_modulos_total`, `num_modulos_leidos` y `porc_modulos_leidos` se obtienen de un bloque intermedio denominado `ModulosVisitadosCU` que tiene su origen en `EstCUM`.
- `num_foros_total`, `num_foros_accedidos` y `porc_foros_accedidos` se obtienen de un bloque intermedio denominado `ForosVisitadosCU` que tiene su origen en `EstCUF`.
- `num_discusiones_total`, `num_discusiones_accedidas` y `porc_discusiones_accedidas` se obtienen de un bloque intermedio denominado `DiscusionesVisitadasCU` que tiene su origen en `EstCUFD`.

- num_apuntes_total, num_apuntes_leidos y porc_apuntes_leidos se obtienen de un bloque intermedio denominado ApuntesVisitadosCU que tiene su origen en EstCUM.

Estadísticas por curso (EstC, tabla: moodle2_bigdata_results.moodle_estadisticas_acceso_curso). Guarda las estadísticas de acceso a cada curso. Presenta la estructura de la Tabla 118.

moodle2_bigdata_results.moodle_estadisticas_acceso_curso		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
num_accesos_totales	int	número de accesos totales, de cualquier tipo que el total de usuarios ha realizado al curso
num_accesos_únicos	int	número de accesos únicos realizados por el total de usuarios al curso. El acceso único se mide al día, es decir, si un usuario realiza varias veces la misma acción el mismo día, solo cuenta una vez. Es una medida del número de cosas distintas que el usuario ha hecho al día
num_accesos_course_view	int	número de accesos realizados por el total de usuarios al página principal del curso
num_accesos_modulos	int	número de accesos realizados por el total de usuarios los módulos con el objetivo de leer su contenido
num_accesos_fuera	int	número de accesos realizados por el total de usuarios desde fuera de la universidad, es decir, aquellos cuya IP de origen no está dentro del conjunto de IPs proporcionadas a la Universidad de Valladolid
num_accesos_dentro	int	número de accesos realizados por el total de usuarios desde dentro de la universidad, es decir, aquellos cuya IP de origen está dentro del conjunto de IPs proporcionadas a la Universidad de Valladolid
porc_accesos_fuera	double	porcentaje de accesos realizados por el total de usuarios desde fuera de la universidad con respecto al total, expresado en tanto por uno
porc_accesos_dentro	double	porcentaje de accesos realizados por el total de usuarios desde dentro de la universidad con respecto al total, expresado en tanto por uno
num_accesos_foros	int	número de visitas realizadas por el total de usuarios a la página principal de los foros publicados en el curso
num_discusiones_creadas	int	número de discusiones creadas por el total de usuarios en cualquiera de los foros del curso
num_accesos_discusiones	int	número de visitas realizadas por el total de usuarios a cualquiera de las discusiones publicadas en el curso
num_post_creados	int	número de post creados por el total de usuarios en cualquiera de las discusiones del curso

num_accesos_apuntes	int	número de accesos realizados por el total de usuarios al total de módulos que contiene información que se puede considerar apuntes (en el código fuente se detalla la elección de estos módulos)
num_accesos_lunes	int	número de accesos realizados por el total de usuarios en días que fueron Lunes
porc_accesos_lunes	double	porcentaje de accesos realizados por el total de usuarios en días que fueron Lunes, en tanto por uno
num_accesos_martes	int	número de accesos realizados por el total de usuarios en días que fueron Martes
porc_accesos_martes	double	porcentaje de accesos realizados por el total de usuarios en días que fueron Martes, en tanto por uno
num_accesos_miercoles	int	número de accesos realizados por el total de usuarios en días que fueron Miércoles
porc_accesos_miercoles	double	porcentaje de accesos realizados por el total de usuarios en días que fueron Miércoles, en tanto por uno
num_accesos_jueves	int	número de accesos realizados por el total de usuarios en días que fueron Jueves
porc_accesos_jueves	double	porcentaje de accesos realizados por el total de usuarios en días que fueron Jueves, en tanto por uno
num_accesos_viernes	int	número de accesos realizados por el total de usuarios en días que fueron Viernes
porc_accesos_viernes	double	porcentaje de accesos realizados por el total de usuarios en días que fueron Viernes, en tanto por uno
num_accesos_sabado	int	número de accesos realizados por el total de usuarios en días que fueron Sábado
porc_accesos_sabado	double	porcentaje de accesos realizados por el total de usuarios en días que fueron Sábado, en tanto por uno
num_accesos_domingo	int	número de accesos realizados por el total de usuarios en días que fueron Domingo
porc_accesos_domingo	double	porcentaje de accesos realizados por el total de usuarios en días que fueron Domingo, en tanto por uno
num_modulos_total	int	número total de módulos publicados en el curso
num_modulos_leidos	int	número total de módulos accedidos o leídos, al menos una vez, por cualquier usuario
porc_modulos_leidos	double	porcentaje de módulos accedidos o leídos, al menos una vez, por cualquier usuario, en tanto por uno
num_foros_total	int	número total de foros publicados en el curso
num_foros_accedidos	int	número total de foros accedidos, al menos una vez, por cualquier usuario

porc_foros_accedidos	double	porcentaje de foros accedidos, al menos una vez, por cualquier usuario, en tanto por uno
num_discusiones_total	int	número total de discusiones publicadas en el curso
num_discusiones_accedidas	int	número total de discusiones accedidas, al menos una vez, por cualquier usuario
porc_discusiones_accedidas	double	porcentaje de discusiones accedidas, al menos una vez, por cualquier usuario, en tanto por uno
num_apuntes_total	int	número total de apuntes publicados en el curso
num_apuntes_leidos	int	número total de apuntes leídos, al menos una vez, por cualquier usuario
porc_apuntes_leidos	double	porcentaje de apuntes leídos, al menos una vez, por cualquier usuario, en tanto por uno.

Tabla 118. Estructura moodle2_bigdata_results.moodle_estadisticas_acceso_curso

En cuanto a la obtención de estas métricas:

- De num_accesos_totales a num_accesos_apuntes y los accesos según día de la semana se obtienen a partir de EstCU.
- num_modulos_total, num_modulos_leidos y porc_modulos_leidos se obtienen de un bloque intermedio denominado ModulosVisitadosC que tiene su origen en EstCM y Modulos.
- num_foros_total, num_foros_accedidos y porc_foros_accedidos se obtienen de un bloque intermedio denominado PorcForosAccedidosC que tiene su origen en EstCUF (a través de otro bloque intermedio denominado PorcAccesosForos) y Foros.
- num_discusiones_total, num_discusiones_accedidas y porc_discusiones_accedidas se obtienen de un bloque intermedio denominado PorcDiscusionesAccedidasC que tiene su origen en EstCUFD y Discusiones.
- num_apuntes_total, num_apuntes_leidos y porc_apuntes_leidos se obtienen de un bloque intermedio denominado ApuntesVisitadosC que tiene su origen en EstCM y Modulos.

4.6.4. Análisis de los resultados académicos

De manera análoga a las estadísticas de acceso, se puede realizar un análisis de los resultados académicos de los alumnos. En este análisis, solo incluiremos a los alumnos, ya que los profesores no serán usuarios evaluables. De entre todos los elementos evaluables, nos vamos a centrar en *assign*, *quiz* y la nota final obtenida en el curso, por dos razones:

- Ser los elementos que se enumeran en los artículos tomados como referencia para crear métricas.
- Ser los elementos más utilizados en los cursos, y por lo tanto los más susceptibles de tener datos.

En concreto, se realizará el análisis de los siguientes conceptos:

- Calificaciones obtenidas por los alumnos en cada uno de los *assign* publicados por curso.
- Fechas de entrega de los *assign*.
- Estadísticas sobre la actividad de cada alumno en los *assign* de cada curso.
- Estadísticas sobre la actividad que los alumnos en total hacen sobre cada *assign*.
- Calificaciones obtenidas por los alumnos en cada uno de los *quiz* publicados por curso junto con el tiempo dedicado en su realización.
- Estadísticas sobre la actividad de cada alumno en los *quiz* de cada curso.
- Estadísticas sobre la actividad que los alumnos en total hacen sobre cada *quiz.*
- Calificaciones finales del curso.

De entre todos estos criterios, los cuatro conjuntos de estadísticas se guardan particionadas por día, para mantener un histórico de los resultados obtenidos.

Con respecto al contenido de este apartado, se considera que la información del código fuente es suficiente para entender la obtención de las métricas, por lo que no se añadirán ejemplos ilustrativos.

La tabla de **calificaciones de *Assign*** (`CalificacionesAssign`, tabla: `moodle2_bigdata_results.moodle_calificaciones_assign`) guarda información sobre la calificación obtenida por cada alumno en cada uno de los *assign* definidos en el curso, suponiendo que todos los alumnos deben entregar todas las tareas. Presenta la estructura de la Tabla 119.

moodle2_bigdata_results.moodle_calificaciones_assign		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
assign	int	identificador de *assign*
usuario	string	identificador de alumno
notasobre10	string	nota sobre 10, en formato numérico, obtenida por el alumno en el *assign*. Si no ha obtenido nota, “Sin calificar”
calificacion	string	etiqueta sobre la calificación, tendrá alguno de los siguientes valores: • “SUSPENSO”, si la nota es inferior a 5. • “APROBADO”, si la no es igual o superior a 5. • “Sin calificar”, si no se obtuvo nota
estado	string	etiqueta sobre el estado de la entrega. Si el alumno ha entregado la tarea, “ENTREGADO”, en caso contrario “NO ENTREGADO”

Tabla 119. Estructura `moodle2_bigdata_results.moodle_calificaciones_assign`

La información de esta tabla se obtiene a partir de `mdl_grade_grades` y `mdl_grade_items`, haciendo uso de `CursoAssignUsuario` para saber qué información buscar.

La tabla de **entregas de *assign*** (`EntregaAssign`, tabla: `moodle2_bigdata_results.moodle_entrega_assign`) se encargará de guardar la fecha en que cada usuario entregó cada *assign* propuesto en el curso. Mantiene la estructura de la Tabla 120.

moodle2_bigdata_results.moodle_entrega_assign		
Campo	**Tipo**	**Descripción**
curso	`string`	identificador de curso
assign	`int`	identificador de *assign*
usuario	`string`	identificador de alumno
fecha_entrega	`string`	fecha en que el alumno entregó el *assign*, en formato YYYY-MM-dd. Si no realizó la entrega o no se encontraron datos, "Sin datos"

Tabla 120. Estructura `moodle2_bigdata_results.moodle_entrega_assign`

La información se obtiene a partir de `mdl_assign_submission` (a través de una versión obtenida previamente en `mdlAssignSubmission`), usando como referencia `CursoAssignUsuario`.

Para cada alumno en cada curso, guardaremos **estadísticas de actividad en *Assign*** (`EstAssign`, tabla: `moodle2_bigdata_results.moodle_estadisticas_assign`). En concreto, se guardará la información presentada en la Tabla 121.

moodle2_bigdata_results.moodle_estadisticas_assign		
Campo	**Tipo**	**Descripción**
curso	`string`	identificador de curso
usuario	`string`	identificador de alumno
num_assign_entregados	`int`	número de *assign* entregados por el alumno
porc_assign_entregados	`double`	porcentaje de *assign* entregados por el alumno, con respecto al total de *assign* propuestos, en tanto por uno
num_assign_evaluados	`int`	número de *assign* que han sido evaluados al alumno
porc_assign_evaluados	`double`	porcentaje de *assign* que han sido evaluados al usuarios, con respecto al número de *assign* que ha entregado, en tanto por uno
num_assign_aprobados	`int`	número de *assign* aprobados por el alumno, es decir, la nota es igual o superior a 5
porc_assign_aprobados	`double`	porcentaje de *assign* aprobados por el alumno, con respecto al número de *assign* que le han evaluado, en tanto por uno
num_assign_suspensos	`int`	número de *assign* suspendidos por el alumno, es decir, la nota es inferior a 5
porc_assign_suspensos	`double`	porcentaje de *assign* suspendidos por el alumno, con respecto al número de *assign* que le han evaluado, en tanto por uno
num_assign_propuestos	`int`	número de *assign* propuestos al alumno

Tabla 121. Estructura `moodle2_bigdata_results.moodle_estadisticas_assign`

La información se obtiene de `CalificacionesAssign`.

Para cada curso, se mantiene información sobre la **actividad sobre cada *Assign*** (`EstCursoAssign`, tabla: `moodle2_bigdata_results.moodle_estadisticas_curso_assign`). En concreto, se guarda la información de la Tabla 122.

moodle2_bigdata_results.moodle_estadisticas_curso_assign		
Campo	**Tipo**	**Descripción**
curso	`string`	identificador de curso
assign	`int`	identificador de *assign*
num_usuarios_total	`int`	número total de alumnos que deben entregar el *assign*
num_assign_entrados	`int`	número de tareas entregadas por los alumnos
porc_assign_entregados	`double`	porcentaje de alumnos que han entregado el *assign*, en tanto por uno
num_assign_evaluados	`int`	número de alumnos que han sido evaluados
porc_assign_evaluados	`int`	porcentaje de alumnos que han sido evaluados, con respecto a los que han entregado la tarea, en tanto por uno

Tabla 122. Estructura `moodle2_bigdata_results.moodle_estadisticas_curso_assign`

La información se completa a partir de `CalificacionesAssign`.

La tabla de **calificaciones de *Quiz*** (`CalificacionesQuiz`, tabla: `moodle2_bigdata_results.moodle_calificaciones_quiz`) guarda información sobre la calificación obtenida por cada alumno en cada uno de los *quiz* definidos en el curso, suponiendo que todos los alumnos deben entregar todos los cuestionarios. Presenta la estructura de la Tabla 123.

moodle2_bigdata_results.moodle_calificaciones_quiz		
Campo	**Tipo**	**Descripción**
curso	`string`	identificador de curso
quiz	`int`	identificador de *quiz*
usuario	`string`	identificador de alumno
notasobre10	`string`	nota sobre 10, en formato numérico, obtenida por el alumno en el *quiz*. Si no ha obtenido nota, “Sin calificar”
calificacion	`string`	etiqueta sobre la calificación, tendrá alguno de los siguientes valores: • “SUSPENSO”, si la nota es inferior a 5. • “APROBADO”, si la no es igual o superior a 5. • “Sin calificar”, si no se obtuvo nota
estado	`string`	etiqueta sobre el estado de la entrega. Si el alumno ha completado el *quiz*, “COMPLETADO”, en caso contrario “NO COMPLETADO”
tiempodedicado	`string`	tiempo dedicado por el alumno para completar el quiz, en segundos. Si el alumno completo el *quiz*, pero no se encontró información, “Sin datos”. Si el alumno no completó el quiz, “No aplicable”

Tabla 123. Estructura `moodle2_bigdata_results.moodle_calificaciones_quiz`

La información de esta tabla se obtiene a partir de `mdl_grade_grades` y `mdl_grade_items`, haciendo uso de `CursoQuizUsuario` para saber qué información buscar.

Para cada alumno en cada curso, guardaremos **estadísticas de actividad en *Quiz*** (`EstQuiz`, tabla: `moodle2_bigdata_results.moodle_estadisticas_quiz`). En concreto, se guardará la información presentada en la Tabla 124.

moodle2_bigdata_results.moodle_estadisticas_quiz		
Campo	**Tipo**	**Descripción**
curso	`string`	identificador de curso
usuario	`string`	identificador de alumno
num_quiz_completados	`int`	número de *quiz* completados por el alumno
porc_quiz_completados	`double`	porcentaje de *quiz* completados por el alumno, con respecto al total de *quiz* propuestos, en tanto por uno
num_quiz_evaluados	`int`	número de *quiz* que han sido evaluados al alumno
porc_quiz_aprobados	`double`	porcentaje de *quiz* aprobados por el alumno, con respecto al número de *quiz* que ha completado, en tanto por uno
num_quiz_suspensos	`int`	número de *quiz* suspendidos por el alumno, es decir, la nota es inferior a 5
porc_quiz_suspensos	`double`	porcentaje de *quiz* suspendidos por el alumno, con respecto al número de *quiz* que ha completado, en tanto por uno
num_assign_propuestos	`int`	número de *quiz* propuestos al alumno

Tabla 124. Estructura `moodle2_bigdata_results.moodle_estadisticas_quiz`

La información se obtiene de `CalificacionesQuiz`.

Para cada curso, se mantiene información sobre la **actividad sobre cada *Quiz*** (`EstCursoQuiz`, tabla: `moodle2_bigdata_results.moodle_estadisticas_curso_quiz`). En concreto, se guarda la información de la Tabla 125.

moodle2_bigdata_results.moodle_estadisticas_curso_quiz		
Campo	**Tipo**	**Descripción**
curso	`string`	identificador de curso
quiz	`int`	identificador de *quiz*
num_usuarios_total	`int`	número total de alumnos que deben entregar el *quiz*
num_quiz_entrados	`int`	número de *quiz* completados por los alumnos
porc_quiz_entregados	`double`	porcentaje de alumnos que han completado el *quiz*, en tanto por uno

Tabla 125. Estructura `moodle2_bigdata_results.moodle_estadisticas_curso_quiz`

La información se completa a partir de `CalificacionesQuiz`.

La tabla de **calificaciones finales** (`CalificacionesFinales`, tabla: `moodle2_bigdata_results.moodle_calificaciones_finales`) almacena la calificación obtenida en el curso por cada alumno, según la estructura de la Tabla 126.

moodle2_bigdata_results.moodle_calificaciones_finales		
Campo	**Tipo**	**Descripción**
curso	`string`	identificador de curso
usuario	`string`	identificador de alumno
notasobre10	`string`	nota sobre 10, en formato numérico, obtenida por el alumno en el curso. Otros supuestos:

	• Si el alumno no ha recibido calificación, pero sigue estando activo en el curso: “Sin calificar”. • Si el alumno se desmatriculó del curso antes de que este acabara y antes de recibir nota, “ABANDONO” • Si el curso acabó sin que el usuario recibiera calificación, “NO CALIFICADO”
calificacion `string`	etiqueta sobre la calificación, tendrá alguno de los siguientes valores: • “SUSPENSO”, si la nota es inferior a 5. • “APROBADO”, si 5 ≤ nota < 7 • “NOTABLE”, si 7 ≤ nota < 9 • “EXCELENTE”, si la nota es igual o superior a 9. • “Sin calificar”, si no se obtuvo nota • “ABANDONO”, si el alumno se desmatriculó del curso antes de que este acabara y antes de recibir nota. • “NO CALIFICADO”, si el curso acabó sin que el alumno recibiera calificación

Tabla 126. Estructura `moodle2_bigdata_results.moodle_calificaciones_finales`

La información se obtiene a partir de `mdl_grade_grades` y `mdl_grade_items`, tomando como referencia la tabla de matriculaciones.

4.6.5. Influencia de la actividad en los resultados académicos

A partir de los dos conjuntos de resultados obtenidos anteriormente, estadísticas de acceso y calificaciones, podemos obtener la influencia que tiene el uno sobre el otro, por ejemplo, siguiendo el artículo [10], tomado como referencia y al que le hemos ido modificando y añadiendo nuevas métricas en función de los datos que teníamos disponibles. Por ejemplo, calcularemos métricas del tipo: qué influencia tiene el número de accesos al total del curso sobre la nota obtenida en él. Esta influencia se calcula por medio de un cálculo de correlación, que en el caso de *Hive*, se proporciona mediante una función que calcula la correlación de *Pearson*.

La correlación entre dos elementos se puede definir como la relación que hay entre ambos, es decir, cómo le afecta a uno el comportamiento del otro. Si la correlación entre dos variables es positiva, implica que cuando una crece la otra también lo hace, de forma más fuerte, cuanto mayor sea la correlación. Si es negativa, cuando una variable crece, la otra desciende. En nuestro caso, tal y como están planteadas las métricas, el comportamiento deseado es que la correlación sea positiva y cuanto más positiva, mejor. Por ejemplo, en el cálculo de la correlación entre el número de accesos totales y la nota final de los alumnos, el comportamiento deseado (como indica la lógica) es que cuantas más visita haga el alumno, mejor sea su nota, y por lo tanto se querrá una correlación lo más positiva posible. La lógica indica que es bastante raro que ocurra lo contrario: cuantas más visitas hagan los alumnos, peor nota obtengan. Esto podría ser un buen indicativo de que existe algún problema en la impartición del curso, ya que, por ejemplo, el acceso a *Moodle* no está sirviendo de ayuda, sino más bien, todo lo contrario.

En cuanto al uso de la función de correlación de *Hive* (`CORR()`), calcularemos la correlación por curso, por lo que para cada curso deberemos generar dos columnas, una por cada una de las dos métricas que queremos relacionar, por ejemplo, una columna para el número de accesos totales y otra para la nota final. Cada una de las entradas de esas columnas representa un alumno de ese curso. Juntando el comportamiento de todos los alumnos de este curso se cal-

cula la relación entre los dos elementos que correspondan, lo cual dará lugar a la correlación en ese curso. Información sobre el cálculo de la correlación de *Pearson* se puede obtener en [205], mientras que en el código fuente de este trabajo se puede consultar los pasos seguidos para aplicar la función `CORR()` de *Hive* en los casos que se presentan a continuación.

En este trabajo, se han calculado las siguientes influencias, agrupadas por similitud:

- Influencia de la actividad en *Moodle* sobre la nota final.
- Influencia del número de accesos a cada módulo sobre:
    - La nota final obtenida en el curso.
    - La nota media en *assign*.
    - La nota media en *quiz*.
    - La nota en cada *assign*.
    - La nota en cada *quiz*.
- Influencia del tiempo dedicado en la realización de un *quiz* sobre la nota en el mismo.
- Influencia del tiempo total dedicado en la realización de *quiz* sobre:
    - La nota media en *quiz*.
    - La nota final obtenida en el curso.

Se resumen a continuación los cálculos realizados.

Influencia de la actividad en *Moodle* sobre la nota final (`CorrAccesosNota`, tabla: `moodle2_bigdata_results.curso_usuario_correlacion`). Mediante esta influencia calculamos, para cada curso, el conjunto de correlaciones mostradas en la Tabla 127, todas relativas a la nota final obtenida en el curso.

`moodle2_bigdata_results.curso_usuario_correlacion`		
Campo	**Tipo**	**Descripción**
`curso`	`string`	identificador de curso
`corr_accesos_totales`	`string`	correlación entre el número de accesos totales al curso y la nota final obtenida en él
`corr_accesos_unicos`	`string`	correlación entre el número de accesos únicos al día al curso y la nota final obtenida en él
`corr_accesos_course_view`	`string`	correlación entre el número de accesos a la página principal del curso y la nota final
`corr_accesos_modulos`	`string`	correlación entre el número de accesos o visitas realizadas a los módulos del curso y la nota final obtenida en él
`corr_accesos_fuera`	`string`	correlación entre el número de accesos al curso realizados desde fuera de la Universidad y la nota final obtenida en él
`corr_acceso_dentro`	`string`	correlación entre el número de accesos al curso realizados desde dentro de la Universidad y la nota final obtenida en él

corr_accesos_foros	string	correlación entre el número de accesos realizados a los foros del curso y la nota final
corr_accesos_discusiones	string	correlación entre el número de accesos realizados a las discusiones publicadas en el curso y la nota final obtenida en él
corr_accesos_apuntes	string	correlación entre el número de visitas a los apuntes del curso y la nota final obtenida en él
corr_accesos_lunes	string	correlación entre el número de accesos realizados en lunes y la nota final obtenida
corr_accesos_martes	string	correlación entre el número de accesos realizados en lunes y la nota final obtenida en el curso
corr_accesos_miercoles	string	correlación entre el número de accesos realizados en miércoles y la nota final obtenida en el curso
corr_accesos_jueves	string	correlación entre el número de accesos realizados en jueves y la nota final obtenida en el curso
corr_accesos_viernes	string	correlación entre el número de accesos realizados en viernes y la nota final obtenida en el curso
corr_accesos_sabado	string	correlación entre el número de accesos realizados en sábado y la nota final obtenida en el curso
corr_accesos_domingo	string	correlación entre el número de accesos realizados en domingo y la nota final obtenida en el curso
corr_modulos_leidos	string	correlación entre el porcentaje de módulos leídos y la nota final obtenida en el curso
corr_foros_accedidos	string	correlación entre el porcentaje de foros accedidos y la nota final obtenida en el curso
corr_discusiones_accedidas	string	correlación entre el porcentaje de foros accedidos y la nota final obtenida en el curso
corr_apuntes_leidos	string	correlación entre el porcentaje de apuntes leídos y la nota final obtenida en el curso
corr_assign_entregados	string	correlación entre el porcentaje de *assign* entregados y la nota final obtenida en el curso
corr_assign_aprobados	string	correlación entre el porcentaje de *assign* aprobados y la nota final obtenida en el curso
corr_assign_suspensos	string	correlación entre el porcentaje de *assign* suspensos y la nota final obtenida en el curso
corr_quiz_completados	string	correlación entre el porcentaje de *quiz* completados y la nota final obtenida en el curso
corr_quiz_aprobados	string	correlación entre el porcentaje de *quiz* aprobados y la nota final obtenida en el curso
corr_quiz_suspensos	string	correlación entre el porcentaje de *quiz* suspensos y la nota final obtenida en el curso
corr_media_acceso_modulos	string	correlación entre la media de acceso a los módulos y la nota final obtenida en el curso

Tabla 127. Estructura `moodle2_bigdata_results.curso_usuario_correlacion`

Influencia del número de accesos a cada módulo sobre la nota final obtenida en el curso (CorrMNota, tabla: `moodle2_bigdata_results.curso_modulo_correlacion`). Sigue la estructura de la Tabla 128.

moodle2_bigdata_results.curso_modulo_correlacion		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
modulo	int	identificador de módulo
corr_accesos_modulo	string	correlación entre el número de accesos a este módulo y la nota final obtenida en el curso

Tabla 128. Estructura `moodle2_bigdata_results.curso_modulo_correlacion`

Influencia del número de accesos a cada módulo sobre la nota media obtenida en *Assign* (CorrMavgA, tabla: `moodle2_bigdata_results.curso_modulo_correlacion_avg_assign`). Sigue la estructura de la Tabla 129.

moodle2_bigdata_results.curso_modulo_correlacion_avg_assign		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
modulo	int	identificador de módulo
corr_accesos_modulo	string	correlación entre el número de accesos a este módulo y la nota media en *assign*

Tabla 129. Estructura `moodle2_bigdata_results.curso_modulo_correlacion_avg_assign`

Influencia del número de accesos a cada módulo sobre la nota media obtenida en *Quiz* (CorrMavgQ, tabla: `moodle2_bigdata_results.curso_modulo_correlacion_avg_quiz`). Sigue la estructura de la Tabla 130.

moodle2_bigdata_results.curso_modulo_correlacion_avg_quiz		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
modulo	int	identificador de módulo
corr_accesos_modulo	string	correlación entre el número de accesos a este módulo y la nota media en *quiz*

Tabla 130. Estructura `moodle2_bigdata_results.curso_modulo_correlacion_avg_quiz`

Influencia del número de accesos a cada módulo sobre la nota obtenida en cada *Assign* (CorrCMA, tabla: `moodle2_bigdata_results.correlacion_curso_modulo_assign`) Sigue la estructura de la Tabla 131.

moodle2_bigdata_results.correlacion_curso_modulo_assign		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
modulo	int	identificador de módulo
assign	int	identificador de *assign*
corr_modulo_assign	string	correlación entre el número de accesos al módulo y la nota en el *assign*

Tabla 131. Estructura `moodle2_bigdata_results.correlacion_curso_modulo_assign`

Influencia del número de accesos a cada módulo sobre la nota obtenida en cada *Quiz* (CorrCMQ, tabla: `moodle2_bigdata_results.correlacion_curso_modulo_quiz`) Sigue la estructura de la Tabla 132.

moodle2_bigdata_results.correlacion_curso_modulo_quiz		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
modulo	int	identificador de módulo
quiz	int	identificador de *quiz*
corr_modulo_assign	string	correlación entre el número de accesos al módulo y la nota en el *quiz*

Tabla 132. Estructura moodle2_bigdata_results.corralcion_curso_modulo_quiz

Influencia del tiempo dedicado en la realización de un *Quiz* sobre la nota en el mismo (CorrTiempoQuiz, tabla: moodle2_bigdata_results.correlacion_tiempo_dedicado_quiz). Sigue la estructura de la Tabla 133.

moodle2_bigdata_results.correlacion_tiempo_dedicado_quiz		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
quiz	int	identificador de *quiz*
corr_tiempo_quiz	string	correlación entre el tiempo dedicado en la realización del *quiz* y la nota obtenida en él

Tabla 133. Estructura moodle2_bigdata_results.correlacion_tiempo_dedicado_quiz

Influencia del tiempo total dedicado en la realización de *Quiz* sobre la nota media en *quiz* y la nota final obtenida en el curso (CorrTiempoQuiz, tabla: moodle2_bigdata_results.correlacion_tiempo_dedicado_nota). Sigue la estructura de la Tabla 134.

moodle2_bigdata_results.correlacion_tiempo_dedicado_nota		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
corr_tiempo_notafinal	string	correlación entre el tiempo dedicado en *quiz* y la nota final obtenida en el curso
corr_tiempo_avgquiz	string	correlación entre el tiempo dedicado en *quiz* y la nota media obtenida en los *quiz* del curso

Tabla 134. Estructura moodle2_bigdata_results.correlacion_tiempo_dedicado_nota

En todos los casos, si no se obtiene información, se completa con "Sin datos". Para cada análisis, se guardan los resultados obtenidos junto con la fecha del análisis, en la partición correspondiente, para generar un histórico de los resultados calculados.

4.6.6. Aplicación de algoritmos de *Machine Learning*

Para finalizar, el análisis se completa con la aplicación de algoritmos de *Machine Learning* para obtener resultados más sofisticados. Siguiendo el artículo [15] tomado como referencia, aplicaremos tres tipos de algoritmos con tres objetivos:

- *Clustering*, para la clasificación de usuarios (tanto alumnos como profesores) en uno de hasta tres posibles *clusters*.
- Obtención de reglas de asociación para descubrir los comportamientos típicos de los alumnos.

- Clasificación de alumnos como mecanismo de predicción de nota de los alumnos sin calificar.

Todas ellas calculadas para cada curso.

Antes de todo ello, debemos realizar un ligero preprocesado de los resultados del bloque de estadísticas y del bloque de calificaciones. En concreto, debemos escoger qué métricas, de todas las calculadas, utilizaremos para aplicar estos algoritmos, siguiendo el artículo citado anteriormente, se tomarán las siguientes (para cada usuario en cada curso):

- Número de discusiones creadas.
- Número de *post* creados.
- Número de módulos leídos.
- Numero de foros accedidos.
- Número de apuntes leídos.
- Número de discusiones accedidas.
- Número de *assign* entregados.
- Número de *assign* aprobados.
- Número de *assign* suspensos.
- Número de *quiz* completados.
- Número de *quiz* aprobados.
- Número de *quiz* suspensos.

Así, en primer lugar, se recopila esta información (que estará presente por varias tablas) en una única, la tabla de **estadísticas de *clustering*** (`EstClustering`, tabla: `moodle2_bigdata_results.moodle_estadisticas_clustering`), ya que contendrá la información que usaremos en el *clustering*. Esta presenta la estructura de la Tabla 135.

`moodle2_bigdata_results.moodle_estadisticas_clustering`		
Campo	**Tipo**	**Descripción**
`curso`	`string`	identificador de curso
`usuario`	`string`	identificador de usuario
`num_discusiones_creadas`	`int`	número discusiones creadas por el usuario en el curso
`num_post_creados`	`int`	número de *post* creados por el usuario en el curso
`num_modulos_leidos`	`int`	número de módulos leídos, al menos una vez, por el usuario
`num_foros_accedidos`	`int`	número de foros accedidos, al menos una vez, por el usuario
`num_apuntes_leidos`	`int`	número de apuntes leídos, al menos una vez, por el usuario
`num_discusiones_accedidas`	`int`	número de discusiones accedidas, al menos una

		vez, por el usuario
num_assign_entregados	int	número de *assign* entregados por el usuario
num_assign_aprobados	int	número de *assign* aprobados por el usuario
num_assign_suspensos	int	número de *assign* suspendidos por el usuario
num_quiz_completados	int	número de *quiz* completados por el usuario
num_quiz_aprobados	int	número de *quiz* aprobados por el usuario
num_quiz_suspensos	int	número de *quiz* suspendidos por el usuario

Tabla 135. Estructura moodle2_bigdata_results.moodle_estadisticas_clustering

Donde:

- num_discusiones_creadas, num_post_creados, num_modulos_leidos, num_foros_accedidos, num_apuntes_leidos y num_discusiones_accedidas se obtiene de las EstCU.
- num_assign_entregados, num_assign_aprobados y num_assign_suspensos se obtienen de EstAssign.
- num_quiz_completados, num_quiz_aprobados y num_quiz_suspensos se obtienen de EstQuiz.

Para la aplicación del *clustering* necesitamos los valores numéricos de cada una de las métricas, pero para aplicar los algoritmos de reglas de asociación y clasificación, necesitamos discretizar estas métricas, para obtener mejores resultados, según como se indica en el artículo. Además, necesitamos añadir la nota final del alumno a la lista de atributos anterior, descartando la información de los profesores. Así, discretizaremos siguiendo la siguiente estrategia:

- Las estadísticas (es decir, todos los atributos menos la nota final) se discretizarán en uno de tres posibles valores: ALTO, MEDIO o BAJO. Para ello, para cada métrica y curso, se tomarán los valores máximo y mínimo y se dividirá el rango entre ambos en tres intervalos iguales:

- En cuanto a la nota final, si tomamos el campo calificación de la tabla de calificaciones finales, este ya viene discretizado de la forma:

Así, se obtiene la tabla de **estadísticas discretizadas** (EstDiscretizadas, tabla: moodle2_bigdata_results.moodle_estadisticas_discretizadas), la cual presenta la estructura de la Tabla 136.

moodle2_bigdata_results.moodle_estadisticas_discretizadas		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
usuario	string	identificador de usuario
num_discusiones_creadas	string	número discusiones creadas por el usuario en el curso
num_post_creados	string	número de *post* creados por el usuario en el curso
num_modulos_leidos	string	número de módulos leídos, al menos una vez, por el usuario
num_foros_accedidos	string	número de foros accedidos, al menos una vez, por el usuario
num_apuntes_leidos	string	número de apuntes leídos, al menos una vez, por el usuario
num_discusiones_accedidas	string	número de discusiones accedidas, al menos una vez, por el usuario
num_assign_entregados	string	número de *assign* entregados por el usuario
num_assign_aprobados	string	número de *assign* aprobados por el usuario
num_assign_suspensos	string	número de *assign* suspendidos por el usuario
num_quiz_completados	string	número de *quiz* completados por el usuario
num_quiz_aprobados	string	número de *quiz* aprobados por el usuario
num_quiz_suspensos	string	número de *quiz* suspendidos por el usuario
nota	string	nota final obtenida por el usuario

Tabla 136. Estructura moodle2_bigdata_results.moodle_estadisticas_discretizadas

Tras ello, ya podemos aplicar los algoritmos de *Machine Learning*. Para ello, haremos uso de las librerías de *Apache Mahout*, las cuales se pueden usar de dos formas:

- A través de línea de comandos que hacen uso de funciones Java.
- A través de programas *Java* que usan esas mismas funciones.

En nuestro caso, hemos preparado una serie de programas ejecutables *Java*, perfectamente configurados, para aplicar los tres algoritmos. Cada uno de ellos hace uso de las librerías *Java* de *Apache Mahout* correspondientes. Se ha notado que no todas las versiones de *Apache Mahout* presentan funciones para implementar todos los algoritmos, en contra de lo lógico, las nuevas versiones han eliminado alguna función o algoritmo que nos hace falta en nuestro caso, por ello, hemos recurrido a librerías de *Apache Mahout* no localizadas en nuestra infraestructura *Hadoop*. Por defecto, no viene instalada ninguna, pero se puede instalar el cliente *Mahout* a través de la interfaz de administrador de *Ambari*. Esto creará, en el directorio de instalación de *Mahout* `/usr/hdp/{versión}/mahout` las librerías correspondientes, las cuales deberán ser exportadas a *HDFS* (o donde corresponda) para la ejecución mediante un programa *Java*. En nuestro caso, hemos añadido una nueva librería descargada del código fuente de *Apache Mahout*, para poder conseguir nuestros objetivos. Se detalla a continuación.

Antes de comenzar a describir el funcionamiento de los programas, conviene aclarar que ninguno de ellos ha sido programado para recibir parámetros de entrada, es decir, todas sus opciones de ejecución están fijadas por código y para cambiarlas hay que recurrir al código y volver a compilar la aplicación. Esto se ha debido a que son varios los parámetros que hay que fijar y para ello se debe conocer qué implican exactamente y qué valores son los adecuados. Programar una etapa previa de comprobación de estos valores para evitar un mal uso de los

algoritmos (que pueda dar lugar a fallos de ejecución) se plantea muy laboriosa ya que los algoritmos son muy sensibles a cambios en sus opciones. Por ello, se ha preferido dejar los algoritmos cerrados, sin opción de cambio de opciones de ejecución, ya que no se sabe si la persona que los va a manejar tendrá conocimientos sobre el uso de ellos y así evitar problemas. Los valores que se han fijado para los parámetros, se han obtenido de otros estudios similares, como el artículo tomado como referencia, de otras fuentes tomadas como referencia para la elaboración del código (referenciadas en el código fuente de las aplicaciones) o aplicando la lógica sobre la definición de cada atributo. Puede ser un punto de mejora de este trabajo, configurar estos paquetes de aplicaciones para que se pueda parametrizar la ejecución de los algoritmos.

4.6.6.1. Clustering: agrupación de usuarios

A partir de las `EstClustering` podemos aplicar un *clustering* (apartado 2.1.1) de usuarios. *Apache Mahout* nos proporciona muchos algoritmos de *clustering*, de entre los cuales, se ha escogido el algoritmo *K – Means* (apartado 2.1.1), por ser el utilizado en el artículo referencia y por ser bastante sencillo de entender y configurar.

Los detalles de programación se pueden encontrar en el código fuente (realizado en base a [206], [207]), pero podríamos resumir lo siguiente (ilustrado en la Figura 96):

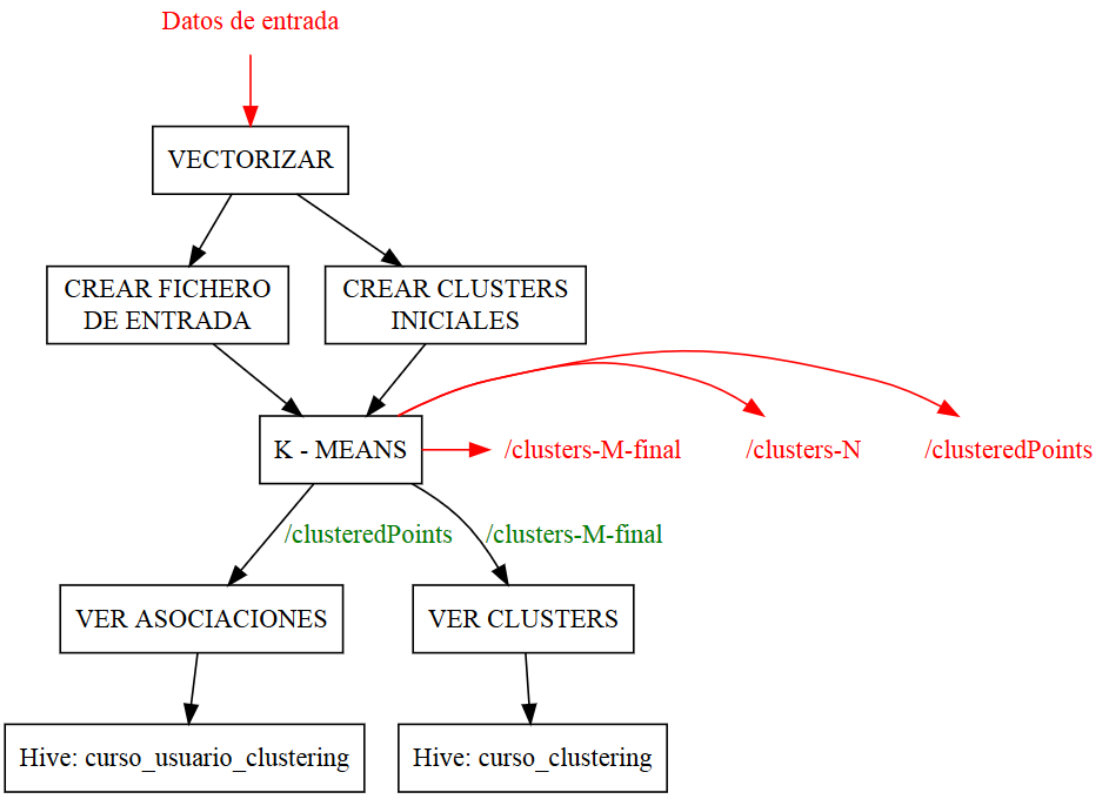


Figura 96. Esquema de procesado de la etapa de *clustering*

- Contará con una clase principal, denominada `es.uva.eduvalab.mahout.clustering.CargaDatos` que se encarga de acceder a *Hive* para obtener los datos de la tabla de `EstClustering`, y generar con ellos una

matriz de datos para pasárselos al algoritmo, implementado en la clase `es.uva.eduvalab.mahout.clustering.Clustering`.

- Una vez ya en el algoritmo, en primer lugar, se deben vectorizar los datos de entrada. Es decir, cada conjunto de parámetros, se convierte en un punto en el espacio vectorial de las coordenadas que correspondan.

- Después, se guardan los datos vectorizados en un fichero en *HDFS*.

- Opcionalmente, se pueden construir los *clusters* iniciales, ya que el algoritmo de *K – Means* necesita partir de unos valores iniciales. Si queremos construir K *clusters*, debemos construir K *clusters* iniciales, seleccionando de la forma que se decida, K puntos de entre los iniciales, por ejemplo, de forma aleatoria.

- Cuando se tiene la información anterior se aplica el algoritmo de *K – Means*, al que le debemos indicar:

    - El directorio en *HDFS* dónde están los datos de entrada.

    - El directorio en *HDFS* donde escribir los resultados.

    - El tipo de distancia utilizada para evaluar los *clusters*.

    - El directorio donde se encuentran los *clusters* iniciales.

    - El número de *clusters* a diferenciar (k). Si se especifica esta opción, se sobrescribe la información del directorio de *clusters* iniciales, ya que se seleccionarán k puntos de los datos de entrada para conformar los *clusters* iniciales. Si no se especifica el número de *clusters*, a partir de la información del directorio de *clusters*, ya se sabe cuántos grupos se van a hacer. En nuestro caso se han escogido diferenciar 3 *clusters*, siguiendo el artículo de referencia.

    - Número máximo de iteraciones.

    - Parámetro Delta de convergencia para decidir si se ha alcanzado resultado final sin cambios tras iterar de nuevo sobre él.

- Como resultado del *clustering*, en el directorio de salida, obtendremos tres tipos de directorios:

    - `clusteredPoints`: contiene las asociaciones entre los puntos iniciales y el *cluster*.

    - `cluster-N`: resultados intermedios en la iteración N.

    - `cluster-M-final`: resultados en la última iteración, M, entre otros contiene las características de los *clusters* obtenidos.

- Con esta información generaremos dos resultados:

    - Asociaciones usuario – *cluster*, la cual se guardará en una tabla denominada `moodle2_bigdata_results.curso_usuario_clustering` (Tabla 137).

    - Descripción de los *clusters*, a partir del cual se generará una segunda tabla denominada `moodle2_bigdata_results.curso_clustering` (Tabla 138).

moodle2_bigdata_results.curso_usuario_clustering		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
usuario	string	identificador de usuario
cluster	string	identificador del *cluster* en el que se ha encuadrado al usuario

Tabla 137. Estructura moodle2_bigdata_results.curso_usuario_clustering

moodle2_bigdata_results.curso_clustering		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
cluster	string	identificador de un *cluster* obtenido para el curso
descripcion	string	descripción del *cluster*

Tabla 138. Estructura moodle2_bigdata_results.curso_clustering

Toda esta aplicación se engloba bajo un bloque denominado `Clustering` que se encargará de:

- Definir (carga inicial) o actualizar (carga incremental) las tablas *Hive* para los resultados.
- Aplicar el programa *Java* de *clustering* de usuarios.

Para ejecutar este programa se necesitan las siguientes librerías, las cuales se pueden encontrar instaladas en la *sandbox* de *Hortonworks*, aunque no se garantiza un correcto funcionamiento si la versión es distinta.

- `aws-java-sdk-1.7.4.jar`
- `hadoop-common-2.7.1.2.3.0.0-2557.jar`
- `hadoop-nfs-2.7.1.2.3.0.0-2557.jar`
- `hadoop-annotations-2.7.1.2.3.0.0-2557.jar`
- `hadoop-auth-2.7.1.2.3.0.0-2557.jar`
- `hadoop-aws-2.7.1.2.3.0.0-2557.jar`
- `mahout-mrlegacy-0.9.0.2.3.0.0-2557-job.jar`
- `mahout-integration-0.9.0.2.3.0.0-2557.jar`
- `mahout-math-0.9.0.2.3.0.0-2557.jar`
- `hadoop-hdfs-2.7.1.2.3.0.0-2557.jar`
- `hive-jdbc-1.2.1.2.3.0.0-2557-standalone.jar`
- `hive-jdbc-1.2.1.2.3.0.0-2557.jar`
- `hive-jdbc.jar`
- `hive-exec-1.2.1.2.3.0.0-2557.jar`

4.6.6.2. Reglas de asociación: análisis del comportamiento de los alumnos

Cuando ya tengamos las estadísticas discretizadas podemos aplicar el algoritmo de reglas de asociación (apartado 2.1.1). En este caso, solo lo haremos sobre los alumnos del curso. Se ha decidido así por simplicidad, ya que esta etapa y la de clasificación, comparten el origen de los datos, `EstDiscretizadas`, y la clasificación no puede contener información de profesores, ya que falsearía las predicciones de nota, si aplicamos los criterios de calificación programados. Así, mantenemos en `EstDiscretizadas` solo a los alumnos, y buscamos las reglas de asociación solo sobre el comportamiento de estos, que por otro lado, es donde está la información interesante e importante.

Como algoritmo de Reglas de Asociación utilizaremos *FP Growth* ya que es la única opción que nos permite *Apache Mahout*, y ni siquiera en las versiones nuevas; hemos tenido que recurrir a una librería descargada del código fuente de *Mahout*, ya que no se instala en *Hortonworks*. El hecho de utilizar *FP Growth* nos obliga a una segunda etapa de preprocesado a partir de las `EstDiscretizadas`.

Los detalles de programación se pueden encontrar en el código fuente, pero podríamos resumir lo siguiente (realizado en base a [18], [206], [208]), ilustrado en la Figura 97:

- Contará con una clase principal, denominada `es.uva.eduvalab.mahout.reglasasociacion.CargaDatos` que se encarga de acceder a *Hive* para obtener los datos de la tabla de `EstDiscretizadas`, y generar con ellos una matriz de datos para pasárselos al algoritmo, implementado en la clase `es.uva.eduvalab.mahout.reglasasociaciones.ReglasAsociacion`, el cual se repite para cada curso.

- Antes de llegar al algoritmo, los datos se deben volver a transformar, no valen con que estén discretizados, sino que también hay que convertirlos a *booleanos*. Esto está impuesto por la forma de funcionamiento del algoritmo *FP Growth*. Para cada transacción (recordar el apartado 2.1.1), en nuestro caso un usuario, debemos indicarle solo los atributos que presenta, es decir, supongamos que se pueden tener dos atributos A y B, y un usuario solo tiene A, a *FP Growth* se le debe indicar que el usuario tiene A; pero si otro usuario tiene los dos, le debemos indicar (A, B). Esto en primer lugar, nos obliga a triplicar el número de atributos, ya que, por ejemplo, de `num_modulos_leidos` tendríamos que diferenciar: `num_modulos_leidos=BAJO`, `num_modulos_leidos=MEDIO` y `num_modulos_leidos=ALTO`, y poner a "*true*" aquel de los tres que tiene el usuario. Por ejemplo, si respectivamente los denominados A, B y C y tenemos dos usuarios, el primero con este parámetro a ALTO y el segundo a MEDIO; a *FP Growth* le diríamos que el primero tiene el atributo C y el segundo B.

- Hecha esta transformación, debemos escribir los resultados en un fichero en *HDFS*, de donde debemos obtener el número de transacciones hechas, simplemente, el número de usuarios que se están analizando, ya que se necesitará más adelante.

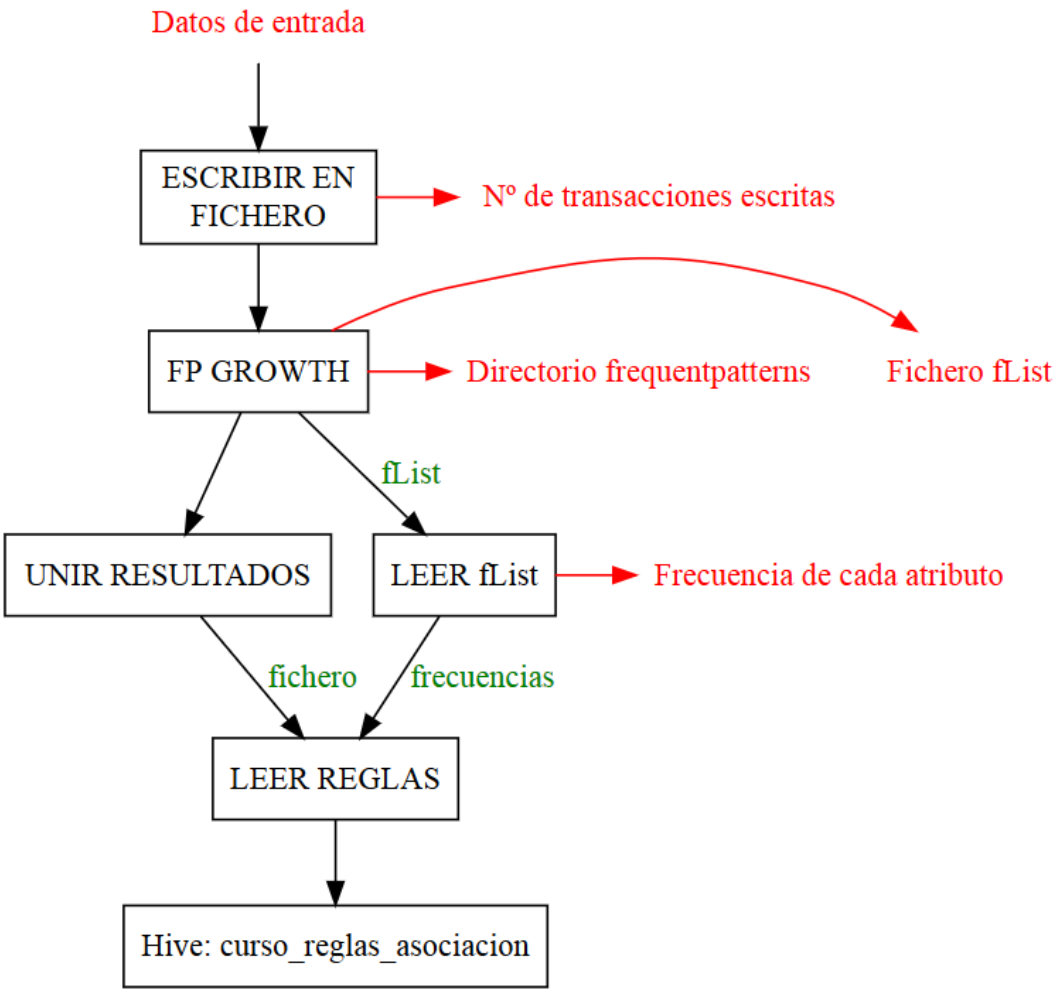


Figura 97. Esquema de procesado de la etapa de Reglas de asociación

- Con los datos ya preparados se aplica el algoritmo de *FP Growth*, al que debemos indicar:
    - El directorio donde se encuentran los datos de entrada.
    - El directorio donde guardar los resultados obtenidos.
    - Número de reglas de asociación buscadas para cada ítem, en nuestro código 10. Un ítem se entiende como cada atributo, por ejemplo un ítem es `num_modulos_leidos=ALTO`. Le estamos diciendo que para este atributo busquemos las 10 reglas más populares o frecuentes que contienen `num_modulos_leidos=ALTO` como consecuente.
    - Método de ejecución *MapReduce*.
    - Número mínimo de transacciones (usuarios) en las que debe estar presente cada ítem/atributo para ser considerado. En este caso vamos a hacer un análisis muy minucioso, ya que hemos considerado un mínimo de 2 transacciones. Es decir, que para analizar `num_modulos_leidos=ALTO` como consecuente de las reglas, debe estar presente en al menos dos usuarios, para tratar de encontrar cualquier comportamiento marginal. Se ignora el caso en que solo esté

presente en un usuario, ya que la cantidad de reglas generadas puede ser enorme (ya lo es con un mínimo de dos).

- Como resultado de la aplicación del algoritmo, obtendremos, entre otros:
    - Un fichero denominado `fList` que contiene el número de transacciones en las que aparece cada ítem/atributo.
    - Un directorio denominado `frecuentpatterns` que contendrá uno o varios ficheros con las reglas generadas.
- Para poder analizar los resultados, necesitamos unir los ficheros del directorio `frequentpatterns` en uno solo.
- Para poder analizar las reglas leemos el contenido de `fList` para saber la frecuencia de aparición de cada ítem, valor necesario para calcular el soporte (*support*) y la confianza (*confidence*) de la regla.
- De entre todas las reglas en `frequentpatterns` nos quedamos con las que cumplan ciertos criterios:
    - Su soporte sea superior a un cierto límite, en nuestro caso 0.3, es decir, la regla debe aparecer en el 30 % del total de transacciones para ser considerada como buena o frecuente.

$$support_{x \to y} = \frac{número\ de\ transacciones\ que\ contienen\ X\ e\ Y}{número\ total\ de\ transacciones}$$

    - Su confianza sea superior a un cierto límite, en nuestro caso 0.4, es decir, los atributos que conforman la regla deben aparecer en el 40% del total de transacciones en las que aparece el consecuente de la regla.

$$confidence_{x \to y} = \frac{support_{x \to y}}{support_y} = \frac{\frac{número\ de\ transacciones\ que\ contienen\ X\ e\ Y}{número\ total\ de\ transacciones}}{\frac{número\ de\ transacciones\ que\ contienen\ Y}{número\ total\ de\ transacciones}} = \frac{número\ de\ transacciones\ que\ contienen\ X\ e\ Y}{número\ total\ de\ transacciones\ que\ contienen\ Y}$$

En la práctica, tras analizar varios ejemplos, se ha comprobado que realmente existe una variación en el cálculo de *confidence* con respecto a la definición dada en el apartado 2.1.1, dados los valores que podemos obtener del algoritmo. En el caso teórico se calcula como la relación entre el soporte de la regla y el soporte del antecedente, mientras que en la práctica se utiliza el soporte del consecuente, ya que si el antecedente está compuesto por varios atributos, no hay forma de obtener esta información con los resultados que nos da el algoritmo *FP Growth* de *Apache Mahout*.

Ambos valores se han fijado aplicando lógica para generar un número manejable de reglas, ya que los valores utilizados en los ejemplos consultados, eran demasiados bajos, ambos por debajo del 0.1. La decisión por escoger los valores para soporte y confianza indicados anteriormente se basa en cuestiones prácticas dados los datos con los que se probaron los algoritmos. Obviamente, se deberá analizar cuáles son los valores adecuados para fijar unos umbrales coherentes. En el caso de este trabajo, estos umbrales, nos permitían generan una cantidad de reglas manejable para poder probar los mecanismos implementados.

- Una vez seleccionadas las reglas, se escriben en *Hive* en una tabla denominada `moodle2_bigdata_results.curso_reglas_asociacion`, la cual sigue la estructura de la Tabla 139.

moodle2_bigdata_results.curso_reglas_asociacion		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
consecuente	string	consecuente de la regla de asociación
regla	string	regla de asociación, contiene el *support* y *confidence* que la caracteriza[7].

Tabla 139. Estructura `moodle2_bigdata_results.curso_reglas_asociacion`

Toda esta aplicación se engloba bajo un bloque denominado `ReglasAsociacion` que se encargará de:

- Definir (carga inicial) o actualizar (carga incremental) las tablas *Hive* para los resultados.
- Aplicar el programa *Java* de Reglas de Asociación.

Para ejecutar este programa se necesitan las siguientes librerías, las cuales se pueden encontrar instaladas en la *sandbox* de *Hortonworks*, aunque no se garantiza un correcto funcionamiento si la versión es distinta.

- `mahout-core-0.9.jar` (obtenida del código fuente de *Mahout*).
- `aws-java-sdk-1.7.4.jar`
- `hadoop-common-2.7.1.2.3.0.0-2557.jar`
- `hadoop-nfs-2.7.1.2.3.0.0-2557.jar`
- `hadoop-annotations-2.7.1.2.3.0.0-2557.jar`
- `hadoop-auth-2.7.1.2.3.0.0-2557.jar`
- `hadoop-aws-2.7.1.2.3.0.0-2557.jar`
- `mahout-mrlegacy-0.9.0.2.3.0.0-2557-job.jar`
- `mahout-integration-0.9.0.2.3.0.0-2557.jar`
- `mahout-math-0.9.0.2.3.0.0-2557.jar`
- `hadoop-hdfs-2.7.1.2.3.0.0-2557.jar`
- `hive-jdbc-1.2.1.2.3.0.0-2557-standalone.jar`
- `hive-jdbc-1.2.1.2.3.0.0-2557.jar`
- `hive-jdbc.jar`
- `log4j-1.2.17.jar`

[7] Por simplicidad, se ha optado por incluir toda las características de la regla en este campo. Un punto de primera mejora del algoritmo puede ser mantener soporte y confianza en campos independientes para un mejor análisis de las reglas.

4.6.6.3. Clasificación de alumnos: predicción de notas

Por último, aplicamos una etapa de clasificación de alumnos con el objetivo de predecir su nota. Atendiendo al funcionamiento de los algoritmos de clasificación del apartado 2.1.1, esta etapa, contará con un procesamiento previo mayor. Vamos a completarla usando como algoritmo *Random Forest*, el único algoritmo de clasificación mediante árboles de decisión disponible en *Mahout*. La única diferencia con los algoritmos más típicos está en que este no generará un árbol, si no varios (el número que le indiquemos) y hará pasar la decisión por todos ellos, para quedarse con la decisión más común.

Partimos de las estadísticas discretizadas. En `EstDiscretizadas` solo guardamos la información de los cursos actuales en *Moodle*, ya que son los únicos sobre los que se recalculan los dos resultados *Machine Learning* anteriores (en los antiguos no habrá cambios). Pero para aplicar clasificación necesitamos información de cursos anteriores, por lo que necesitamos una segunda tabla de histórico de estadísticas discretizadas, igual que `EstDiscretizadas`, pero con información sobre cursos no actuales. Así, en cada análisis tendremos:

- Estadísticas discretizadas, con información sobre los cursos actuales.
- Histórico de estadísticas discretizadas, con información simultánea de cursos ya no existentes en *Moodle* y cursos actuales (es decir, también guarda la misma información que `EstDiscretizadas`). Los cursos no actuales, no pueden tener alumnos con calificación "Sin calificar", ya que esta etiqueta se reserva para usuarios no calificados pero susceptibles de serlo, es decir, están presentes en un curso actualmente en *Moodle*. Recordando la definición de la etiqueta de calificación en la tabla de calificaciones Tabla 126, si un curso acaba, los usuarios "Sin calificar" pasan a "NO CALIFICADO".

Para predecir la nota de los alumnos, se usa información sobre qué ocurrió con alumnos del mismo curso o asignatura, pero en años anteriores (o del mismo año si ya hay alguno calificado). En definitiva, entra en juego la estructura del identificador *BigData* de cursos. Dados dos cursos `curso1-2014` y `curso1-2015`, por su nombre, son la misma asignatura, pero en dos años académicos distintos. La etapa de clasificación podrá juntar la información de ambos cursos para predecir la nota de alumnos sin calificar.

Con estas cuestiones, se generan dos conjuntos de datos, necesarios para la clasificación (apartado 2.1.1):

- **Datos de entrenamiento**. Formados por alumnos ya calificados, las etiquetas "ABANDONO" y "NO CALIFICADO" se consideran alumnos calificados. Estos datos juntarán bajo un mismo identificador de curso, todos los usuarios de la misma asignatura, pero en distintos años. Por ejemplo, los usuarios calificados de los cursos `curso1-2014` y `curso1-2015` conformarán los datos de entrenamiento del curso `curso1`. Se obtiene a partir del histórico de estadísticas discretizadas, descartando los alumnos "Sin calificar".
- **Datos de test o prueba**. Formados por los alumnos aun por calificar, es decir, con calificación "Sin calificar" en la tabla de estadísticas discretizadas, la que contiene solo los cursos actuales. Se predecirá la nota de estos alumnos, en base a los alumnos que forman los datos de entrenamiento. Los datos de prueba se guardan tanto con el identificador de curso que contiene el año académico, como el que no. Por ejemplo, los usuarios del curso `curso1-2014` se guardan con identificador de curso `curso1-2014` e identificador histórico `curso1`, para saber a qué conjunto de datos de entrenamiento hay que recurrir.

Con todos estos datos, ya podemos aplicar el programa *Java* de Clasificación. Los detalles de programación se pueden encontrar en el código fuente (realizado en base a [206], [209]–[212]), pero podríamos resumir lo siguiente, ilustrado en la Figura 98:

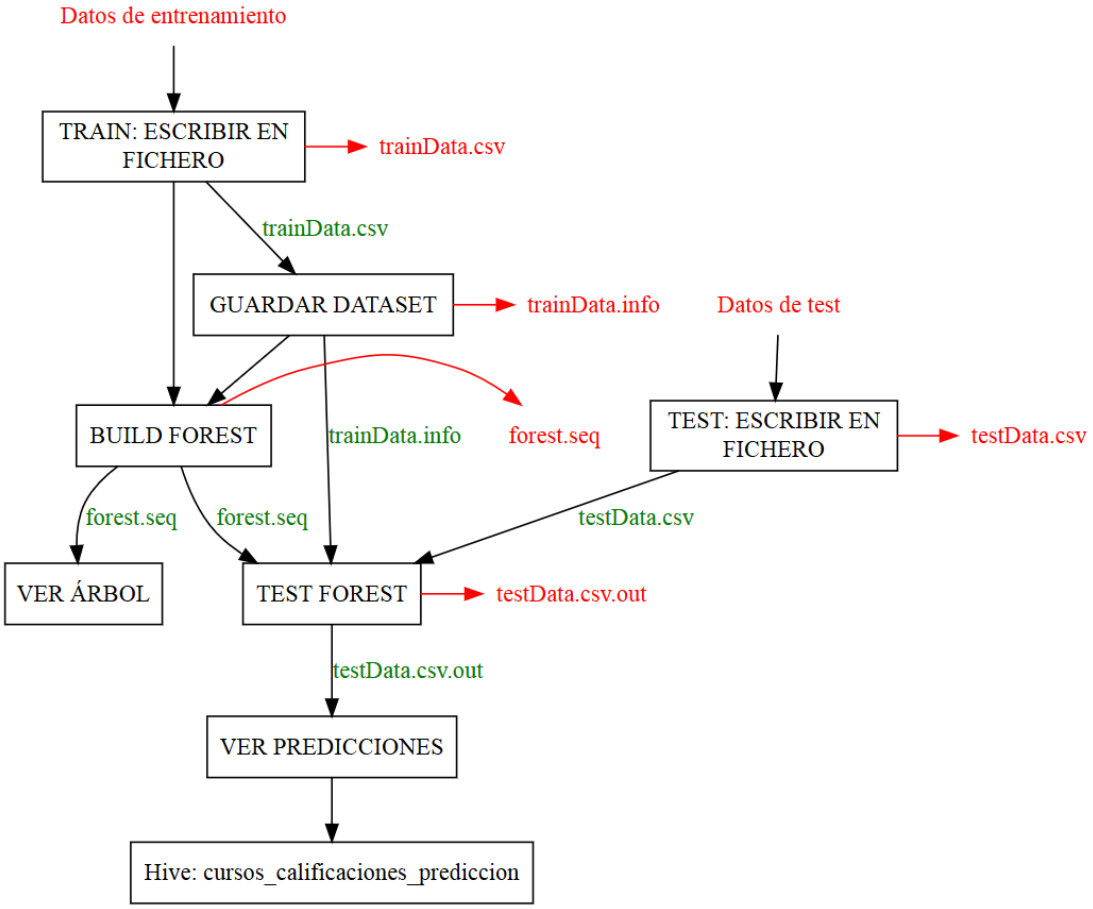


Figura 98. Esquema de procesado de la etapa de clasificación

- Contará con una clase principal, denominada `es.uva.eduvalab.mahout.clasificacion.CargaDatos` que se encarga de acceder a *Hive* para obtener los datos de las tablas correspondientes, y generar con ellos una matriz de datos de datos de entrenamiento y otra de datos de prueba. Se pasarán estos datos al algoritmo, implementado en la clase `es.uva.eduvalab.mahout.clasificacion.Clasificacion`, el cual se repite para cada curso, siempre que haya alumnos a los que predecir la nota y alumnos que usar como referencia.

    - Accedemos a las estadísticas discretizadas para saber para qué cursos tenemos que aplicar el algoritmo, obteniendo relaciones del tipo identificador de `curso -> identificador histórico`. Así sabemos a cada curso actual, con qué identificador histórico se deben buscar los datos de entrenamiento en la tabla *Hive* correspondiente.

    - Cuando se está analizando un curso, sus datos de entrenamiento se buscan con su identificador histórico, es decir, sin la parte que diferencia el año académico del curso. Por ejemplo, para el curso `curso1-2015`, buscaríamos los datos de entrenamiento con identificador `curso1`.

  - Los datos de prueba se buscan con su identificador de curso completo, con año académico, en la tabla de estadísticas discretizadas. Por ejemplo, `curso1-2015`.

- Cuando los datos ya están preparados, se pasa a la ejecución del algoritmo. En primer lugar, se deben guardar ambos conjuntos de datos en los correspondientes ficheros *CSV* en *HDFS*.

- A partir de los datos de entrenamiento se genera el *DataSet*, una descripción de las características de los datos de entrenamiento. Principalmente, qué atributos presenta, cuáles se usan para la clasificación, cuál es la etiqueta que debe ser final de árbol, es decir, cual es el punto final de la clasificación. Para ello debemos indicar también un descriptor de los datos, en el que especificamos las características de cada atributo, pudiendo ser:

  - Parámetro ignorado, no teniendo en cuenta en la clasificación (“I”).

  - Parámetro categórico (“C”).

  - Parámetro numérico (“N”).

  - Etiqueta de la clasificación (“L”).

  Por ejemplo, en nuestro caso, suponiendo el orden de atributos indicados al inicio del Apartado 4.6.6, y teniendo el identificador de usuario en primer lugar, el descriptor tendría la forma:

  ```
  "I C C C C C C C C C C C L"
  ```

  El primer parámetro debe ser ignorado en la clasificación, ya que es el identificador de usuario. Los siguientes doce parámetros, son los atributos en función de los cuales hacer la clasificación, son todos categóricos, ya que solo pueden tomar uno de tres valores (BAJO, MEDIO o ALTO). El último parámetro es la nota del alumno, la etiqueta que debemos predecir, en función de la cual realizar la clasificación. A mayores, el *DataSet* nos devolverá los distintos valores que puede tener la etiqueta, por ejemplo, si entre los datos de entrenamiento solo se han distinguido como nota final “APROBADO” y “NOTABLE” nos devolverá que son posibles estos dos valores para la etiqueta seleccionada.

- A partir del *DataSet* y los datos de entrenamiento se ejecuta la construcción del árbol de decisión. Debemos indicar los siguientes parámetros:

  - El directorio que contiene los datos de entrenamiento.

  - El fichero que contiene el *DataSet.*

  - Número de árboles de decisión a generar. Se ha configurado generar 100 árboles.

  - El directorio donde escribir los resultados.

- Como resultado se genera un fichero `forest.seq` con los árboles generados.

- Escribimos por pantalla estos árboles, a modo de información para el usuario.

- Con los árboles definidos en `forest.seq` y los datos de prueba, predecimos las notas de los alumnos mediante la parte de test del algoritmo, para lo cual indicaremos:
    - El directorio donde se encuentran los datos de prueba a clasificar.
    - El directorio que contiene el *DataSet.*
    - El directorio que contiene a `forest.seq`.
    - El directorio donde guardar los resultados.
- Esto genera un fichero con los resultados de la **predicción**, en concreto con el valor que se le da a la etiqueta de cada uno de los alumnos. Leemos ese resultado y lo guardamos en *Hive*, en la tabla `moodle2_bigdata_results.cursos_calificaciones_prediccion` (Tabla 140).

moodle2_bigdata_results.cursos_calificaciones_prediccion		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
usuario	string	identificador de alumno
nota	string	nota predicha para el alumno

Tabla 140. Estructura `moodle2_bigdata_results.cursos_calificaciones_prediccion`

- Por otro lado, se guarda un **histórico de predicciones**, es decir, la última predicción realizada para los alumnos justo antes de ser calificados. Esta información se guarda en la tabla `moodle2_bigdata_results.cursos_calificaciones_prediccion_historico` (Tabla 141). En base a la última predicción realizada y a la nota realmente obtenida por el alumno, se puede obtener el acierto del sistema de predicción implementado, guardando los resultados en `moodle2_bigdata_results.prediccion_porcentaje _acierto` (Tabla 142)

moodle2_bigdata_results.cursos_calificaciones_prediccion_historico		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
usuario	string	identificador de alumno
nota	string	última nota predicha para el alumno

Tabla 141. Estructura `moodle2_bigdata_results.cursos_calificaciones_prediccion_historico`

moodle2_bigdata_results.prediccion_porcentaje_acierto		
Campo	**Tipo**	**Descripción**
curso	string	identificador de curso
porc_acierto	double	porcentaje de acierto en la predicción de nota de alumnos en este curso

Tabla 142. Estructura `moodle2_bigdata_results.prediccion_porcentaje_acierto`

Toda esta aplicación se engloba bajo un bloque denominado `Clasificación` que se encargará de:

- Generar el histórico de estadísticas discretizadas.
- Formar los datos de entrenamiento y prueba.
- Definir (carga inicial) o actualizar (carga incremental) las tablas *Hive* para los resultados.

- Aplicar el programa *Java* de Clasificación.
- Calcular el porcentaje de acierto.

Para ejecutar este programa se necesitan las siguientes librerías, las cuales se pueden encontrar instaladas en la *sandbox* de *Hortonworks*, aunque no se garantiza un correcto funcionamiento si la versión es distinta.

- `aws-java-sdk-1.7.4.jar`
- `hadoop-common-2.7.1.2.3.0.0-2557.jar`
- `hadoop-nfs-2.7.1.2.3.0.0-2557.jar`
- `hadoop-annotations-2.7.1.2.3.0.0-2557.jar`
- `hadoop-auth-2.7.1.2.3.0.0-2557.jar`
- `hadoop-aws-2.7.1.2.3.0.0-2557.jar`
- `mahout-mrlegacy-0.9.0.2.3.0.0-2557-job.jar`
- `mahout-integration-0.9.0.2.3.0.0-2557.jar`
- `mahout-math-0.9.0.2.3.0.0-2557.jar`
- `hadoop-hdfs-2.7.1.2.3.0.0-2557.jar`
- `hive-jdbc-1.2.1.2.3.0.0-2557-standalone.jar`
- `hive-jdbc-1.2.1.2.3.0.0-2557.jar`
- `hive-jdbc.jar`
- `mahout-examples-0.9.0.2.3.0.0-2557.jar`

4.7. Análisis de resultados

Para comprobar los resultados obtenidos en el apartado 4.6, se transferirán algunos de estos a una base de datos *Elasticserach* con el objetivo de poder visualizarlos gráficamente mediante *Kibana*. Esta parte del trabajo se utilizará solo a nivel experimental, no será la alternativa que se use para implementar en la práctica la etapa de visualización, por lo que no se entrará en detalles en cuanto a su configuración.

En primer lugar, se indicará brevemente en qué ha consistido la transferencia de datos a *ElasticSearch*. Después, se presentarán ejemplos gráficos de visualizacíon de las métricas mediante *Kibana*.

4.7.1. Transferencia a *ElasticSearch*

La transferencia de información a *ElasticSearch* se realiza en los términos planteados en el apartado 4.1.6, es decir, contará con los siguientes pasos:

- Creación del índice donde almacenar la información en *ElasticSearch*.
- Creación de una tabla externa en *Hive* que conecte con ese índice.
- Inserción de datos en el índice por medio de *Hive*. Las tablas se han creado en una base de datos denominada `moodle_elasticsearch`.

Para probar nuestros resultados, se transferirán a *ElasticSearch* los resultados de los siguientes bloques:

- Tablas Historial:
  - `Cursos`.
  - `Matriculaciones`.
  - `Modulos`.
- Estadísticas de acceso:
  - `EstDCU`.
  - `EstCM`.
  - `EstCUM`.
  - `EstCU`.
  - `EstC`.
- Resultados académicos:
  - `CalificacionesAssign`.
  - `CalificacionesQuiz`.
  - `CalificacionesFinales`.
  - `EstAssign`.

Bloque	Tabla *Hive*	Tabla *Hive – ES*	Índice *ES*
CalificacionesAssign	moodle_ calificacaciones_assign	moodle_calificaciones_assign_es	moodle_calificaciones_assign
CalificacionesFinales	moodle_ calificaciones_finales	moodle_calificaciones_finales_es	moodle_calificaciones_finales
CalificacionesQuiz	moodle_calificaciones_quiz	moodle_calificaciones_quiz_es	moodle_calificaciones_quiz
CorrAccesosNota	curso_usuario_correlacion	curso_usuario_correlacion_es	curso_usuario_correlacion
		curso_usuario_correlacion_ representacion_origen_accesos_es	curso_usuario_correlacion_ representacion_origen_accesos
		curso_usuario_correlacion_ representacion_accesos_es	curso_usuario_correlacion_ representacion_accesos
		curso_usuario_correlacion_ representacion_actividad_es	curso_usuario_correlacion_ representacion_actividad
		curso_usuario_correlacion_ representacion_dia_semana_es	curso_usuario_correlacion_ representacion_dia_semana
		curso_usuario_correlacion_ representacion_item_es	curso_usuario_correlacion_ representacion_item
CorrCMA	correlacion_curso_modulo_assign	correlacion_curso_modulo_assign_es	correlacion_curso_modulo_assign
		correlacion_curso_modulo_ assign_numerico_es	correlacion_curso_modulo_ assign_numerico
CorrCMQ	correlacion_curso_modulo_quiz	correlacion_curso_modulo_quiz_es	correlacion_curso_modulo_quiz
		correlacion_curso_modulo_ quiz_numerico_es	correlacion_curso_modulo_ quiz_numerico
CorrMavgA	curso_modulo_correlacion_avg_assign	curso_modulo_correlacion_ avg_assign_es	curso_modulo_correlacion_ avg_assign
		curso_modulo_correlacion_ avg_assign_numerico_es	curso_modulo_correlacion_ avg_assign_numerico
CorrMavgQ	curso_modulo_correlacion_avg_quiz	curso_modulo_correlacion_ avg_quiz_es	curso_modulo_correlacion_ avg_quiz
		curso_modulo_correlacion_ avg_quiz_numerico_es	curso_modulo_correlacion_ avg_quiz_numerico
CorrMNota	curso_modulo_correlacion	curso_modulo_correlacion_es	curso_modulo_correlacion
		curso_modulo_correlacion_numerico_es	curso_modulo_correlacion_numerico
CorrTiempoQuiz	correlacion_tiempo_dedicado_nota	correlacion_tiempo_ dedicado_nota_es	correlacion_tiempo_ dedicado_nota
		correlacion_tiempo_ dedicado_nota_numerico_es	correlacion_tiempo_ dedicado_nota_numerico
	correlacion_tiempo_dedicado_quiz	correlacion_tiempo_ dedicado_quiz_es	correlacion_tiempo_ dedicado_quiz
		correlacion_tiempo_ dedicado_quiz_numerico_es	correlacion_tiempo_ dedicado_quiz_numerico

CursoClasificacion	cursos_calificaciones_prediccion	curso_prediccion_es	curso_prediccion
	cursos_calificaciones_prediccion_historico	curso_prediccion_historico_es	curso_prediccion_historico
	prediccion_porcentaje_acierto	prediccion_porcentaje_acierto_es	prediccion_porcentaje_acierto
CursoClustering	curso_clustering	curso_clustering_es	curso_clustering
	curso_usuario_clustering	curso_usuario_clustering_es	curso_usuario_clustering
CursoReglasAsociacion	curso_reglas_asociacion	curso_reglas_asociacion_es	curso_reglas_asociacion
Cursos	cursos_historico	moodle_cursos_es	moodle_cursos
EntregaAssign	moodle_entrega_assign	moodle_entrega_assign_es	moodle_entrega_assign
EstAssign	moodle_estadisticas_assign	moodle_estadisticas_assign_es	moodle_estadisticas_assign
		moodle_estadisticas_assign_representacion_es	moodle_estadisticas_assign_representacion
EstCM	moodle_estadisticas_acceso_curso_modulo	moodle_estadisticas_acceso_curso_modulo_es	moodle_estadisticas_acceso_curso_modulo
		moodle_estadisticas_acceso_curso_modulo_representacion_porcentaje_accesos_es	moodle_estadisticas_acceso_curso_modulo_representacion_porcentaje_accesos
EstC	moodle_estadisticas_acceso_curso	moodle_estadisticas_acceso_curso_es	moodle_estadisticas_acceso_curso
		moodle_estadisticas_acceso_curso_representacion_accesos_es	moodle_estadisticas_acceso_curso_representacion_accesos
		moodle_estadisticas_acceso_curso_representacion_dia_semana_es	moodle_estadisticas_acceso_curso_representacion_dia_semana
		moodle_estadisticas_acceso_curso_representacion_discusiones_accedidas_es	moodle_estadisticas_acceso_curso_representacion_discusiones_accedidas
		moodle_estadisticas_acceso_curso_representacion_foros_accedidos_es	moodle_estadisticas_acceso_curso_representacion_foros_accedidos
		moodle_estadisticas_acceso_curso_representacion_origen_accesos_es	moodle_estadisticas_acceso_curso_representacion_origen_accesos
		moodle_estadisticas_acceso_curso_representacion_modulos_leidos_es	moodle_estadisticas_acceso_curso_representacion_modulos_leidos
		moodle_estadisticas_acceso_curso_representacion_apuntes_leidos_es	moodle_estadisticas_acceso_curso_representacion_apuntes_leidos
EstCUM	moodle_estadisticas_acceso_curso_usuario_modulo	moodle_estadisticas_acceso_curso_usuario_modulo_es	moodle_estadisticas_acceso_curso_usuario_modulo
		moodle_estadisticas_acceso_curso_usuario_modulo_representacion_diferencias_es	moodle_estadisticas_acceso_curso_usuario_modulo_representacion_diferencias
EstCU	moodle_estadisticas_acceso_curso_usuario	moodle_estadisticas_acceso_curso_usuario_es	moodle_estadisticas_acceso_curso_usuario
		moodle_estadisticas_acceso_curso_usuario_representacion_accesos_es	moodle_estadisticas_acceso_curso_usuario_representacion_accesos
		moodle_estadisticas_acceso_curso_usuario_representacion_dia_semana_es	moodle_estadisticas_acceso_curso_usuario_representacion_dia_semana

		moodle_estadisticas_acceso_ curso_usuario_representacion_ discusiones_accedidas_es	moodle_estadisticas_acceso_ curso_usuario_representacion_ discusiones_accedidas
		moodle_estadisticas_acceso_ curso_usuario_representacion_ foros_accedidos_es	moodle_estadisticas_acceso_ curso_usuario_representacion_ foros_accedidos
		moodle_estadisticas_acceso_ curso_usuario_representacion_ origen_accesos_es	moodle_estadisticas_acceso_ curso_usuario_ representacion_origen_accesos
		moodle_estadisticas_acceso_ curso_usuario_representacion_ modulos_leidos_es	moodle_estadisticas_acceso_ curso_usuario_representacion_ modulos_leidos/
		moodle_estadisticas_acceso_ curso_usuario_representacion_ apuntes_leidos_es	moodle_estadisticas_acceso_ curso_usuario_representacion_ apuntes_leidos
EstDCU	moodle_estadisticas_ acceso_dia_curso_usuario	moodle_estadisticas_acceso_ dia_curso_usuario_es	moodle_estadisticas_acceso_ dia_curso_usuario
		moodle_estadisticas_acceso_ dia_curso_usuario_representacion_accesos_es	moodle_estadisticas_acceso_ dia_curso_usuario_representacion_accesos
		moodle_estadisticas_acceso_ dia_curso_usuario_ representacion_origen_accesos_es	moodle_estadisticas_acceso_ dia_curso_usuario_ representacion_origen_accesos
EstQuiz	moodle_estadisticas_quiz	moodle_estadisticas_quiz_es	moodle_estadisticas_quiz
		moodle_estadisticas_quiz_ representacion_es	moodle_estadisticas_quiz_ representacion
Matriculaciones	matriculaciones	moodle_matriculaciones_es	moodle_matriculaciones
Modulos	cursos_modulos	moodle_cursos_modulos_es	moodle_cursos_modulos

Tabla 143. Correspondencias *Hive – ElasticSearch*

  - `EstQuiz`.
  - `EntregaAssign`

- Influencia actividad – resultados:
  - `CorrAccesosNota`.
  - `CorrCMA`.
  - `CorrCMQ`.
  - `CorrMNota`.
  - `CorrMavgA`.
  - `CorrMavgQ`.
  - `CorrTiempoQuiz`.
- *Machine Learning*:
  - `CursoClustering`.
  - `CursoClasificacion`.
  - `CursoReglasAsociacion`.

En todos los casos, la transferencia se realiza mediante un bloque con el mismo nombre añadiendo el sufijo `Pres`, por ejemplo, el bloque `Cursos` se transfiere por medio del bloque `CursosPres`.

En la mayor parte de los bloques, la transferencia consistirá en la creación de un índice idéntico a la tabla en *Hive* para crear una copia de la misma. En otros, a mayores, la información de la tabla se ha transferido de distintas formas para ofrecer mayores oportunidades de representación mediante *Kibana*. Para ver los detalles, referimos al lector al código fuente del trabajo. En la Tabla 143, se representa la equivalencia entre tabla *Hive* origen, tabla conexión *Hive* – *ElasticSearch* e índice en *ElasticSearch*.

4.7.2. Representación mediante *Kibana*

Mediante *Kibana* representaremos gráficamente la información transferida a *ElasticSearch*. Estos gráficos se pueden agrupar en paneles (*Dashboard*). A continuación se detallan los paneles definidos así como los gráficos que nos podemos encontrar en ellos. No se detalla su obtención. Junto con el código fuente de este trabajo se proporciona un fichero denominado `KibanaObject.json`. Este es un fichero en formato *JSON* donde se encuentran definidos todos los elementos configurados en *Kibana* para obtener las representaciones que siguen este apartado, a partir de los datos transferidos a *ElasticSearch*. Se puede importar este fichero a cualquier otra aplicación *Kibana* que tenga definidos los índices correspondientes, para poder definir directamente los elementos de visualización aquí empleados. No se ha encontrado ninguna forma de proporcionar una adición automática de los índices *ElasticSearch* por lo que estos deberían ser añadidos manualmente, en base a la Tabla 143.

A continuación se presentan ejemplos de representaciones que se pueden realizar a partir de los resultados del apartado 4.6. Se han generado varios paneles, cada uno de los cuáles cuenta con varias visualizaciones relacionadas bajo un criterio. Se presentan por orden alfabético. En todos estos paneles es necesaria la selección de criterios como curso, módulo, *assign*, *quiz*, usuario, etc., mediante la opción de filtrado de *Kibana*. En las figuras que se presentan a continuación, se puede ver la selección del filtrado en una casilla de búsqueda en la parte superior de la página de *Kibana*. En los casos en que corresponda, todos los porcentajes se representan en tanto por uno.

En el panel **"ANÁLISIS DE UN MÓDULO"** se presenta información sobre la actividad llevada a cabo por los usuarios sobre un módulo. Se puede ver su vista previa en la Figura 99. Se compone de las siguientes visualizaciones.

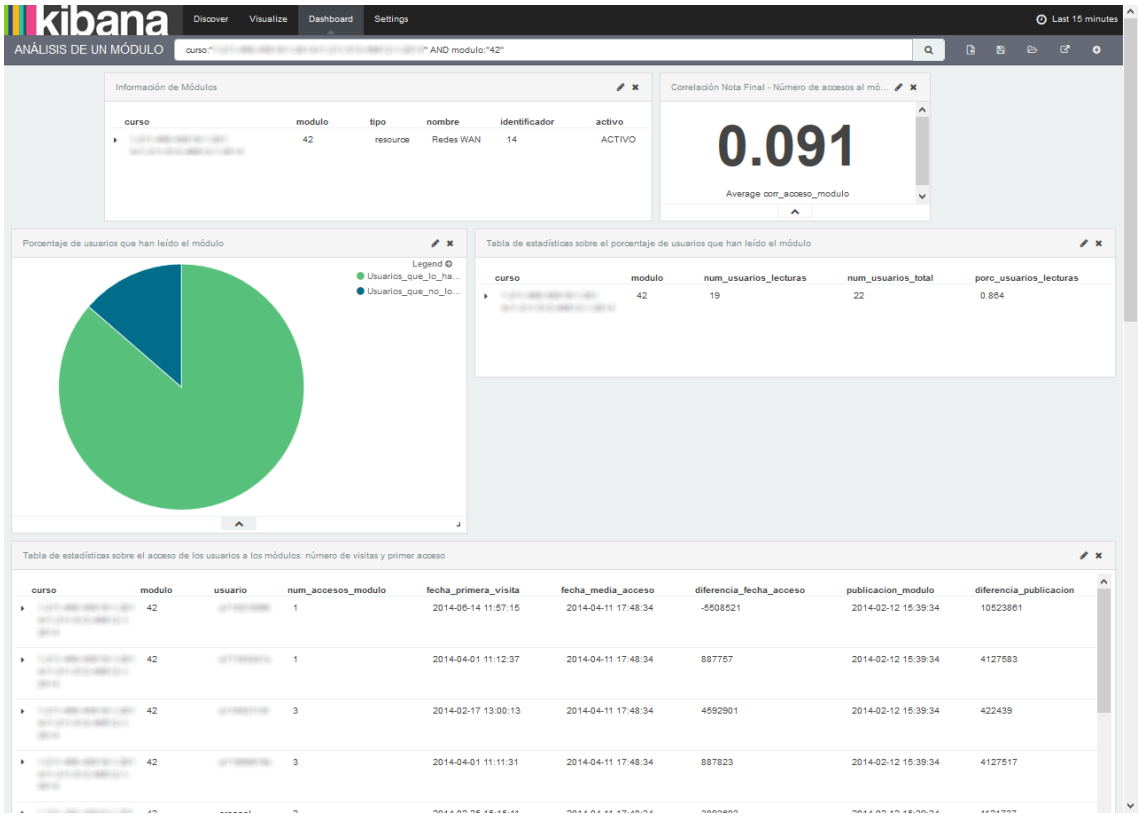


Figura 99. Vista previa del *dashboard* "ANÁLISIS DE UN MÓDULO"

- Información del módulo analizado (Figura 100).

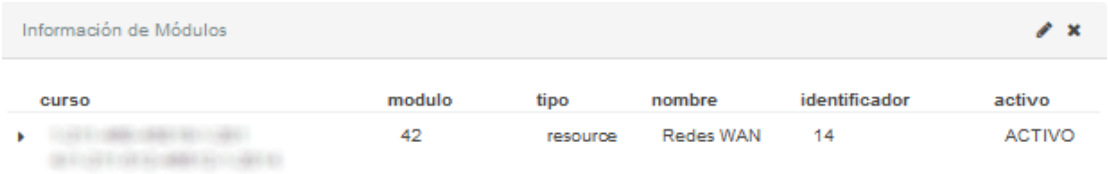


Figura 100. *Kibana*. *Dashboard* "ANÁLISIS DE UN MÓDULO" – Visualización: Información del módulo

- Correlación entre la nota final y el número de accesos al módulo (Figura 101).

Figura 101. *Kibana*. *Dashboard* "ANÁLISIS DE UN MÓDULO" – Visualización: Correlación entre la nota final y el número de accesos al módulo

- Porcentaje de usuarios que han leído el módulo (Figura 102).

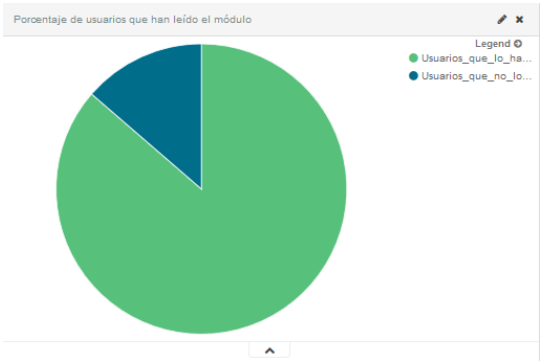


Figura 102. *Kibana*. *Dashboard* "ANÁLISIS DE UN MÓDULO" – Visualización: Porcentaje de usuarios que han leído el módulo

- Tabla de estadísticas sobre el porcentaje de usuarios que han leído el módulo (Figura 103).

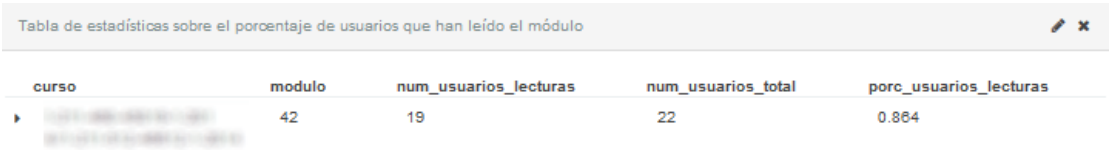


Figura 103. *Kibana*. *Dashboard* "ANÁLISIS DE UN MÓDULO" – Visualización: Tabla de estadísticas sobre el porcentaje de usuarios que han leído el módulo

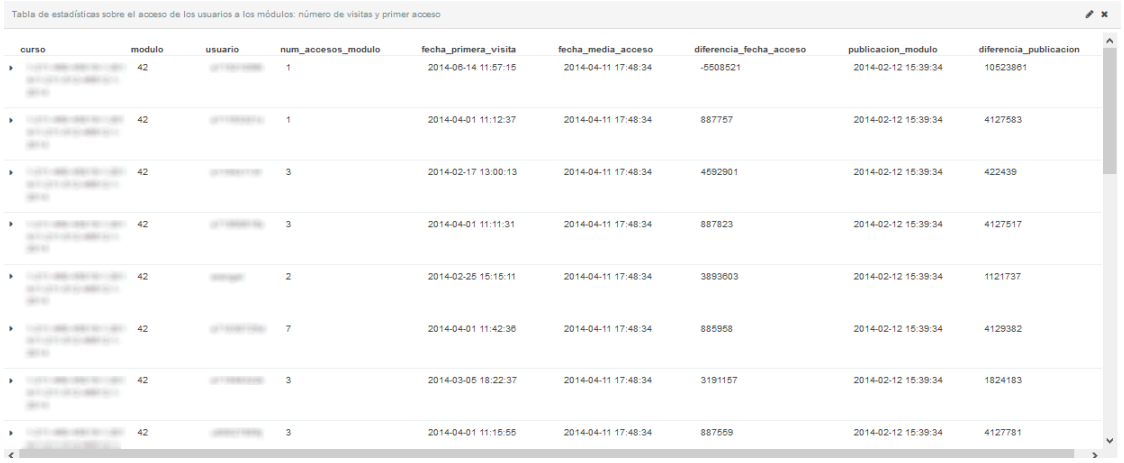


Figura 104. *Kibana*. *Dashboard* "ANÁLISIS DE UN MÓDULO" – Visualización: Tabla de estadísticas sobre el acceso de los usuarios a los módulos

- Tabla de estadísticas sobre el acceso de los usuarios a los módulos, en concreto, número de visitas y fecha del primer acceso junto con el resto de información relacionada obtenida en el apartado 4.6.3 (Figura 104).

- Top 10 usuarios con más accesos al módulo (Figura 105).

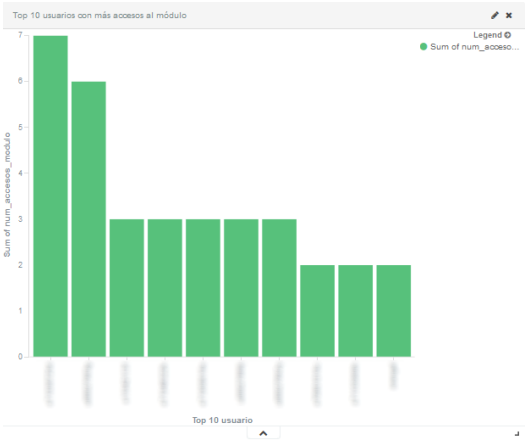


Figura 105. *Kibana*. *Dashboard* "ANÁLISIS DE UN MÓDULO" – Visualización: Top 10 usuarios con más accesos al módulo

- Top 10 usuarios con mayor diferencia en la fecha de acceso con respecto a la publicación del módulo (Figura 106).

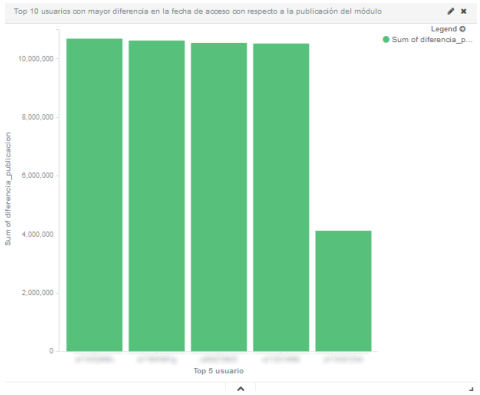


Figura 106. *Kibana*. *Dashboard* "ANÁLISIS DE UN MÓDULO" – Visualización: Top 10 usuarios con mayor diferencia en la fecha de acceso con respecto a la publicación del módulo

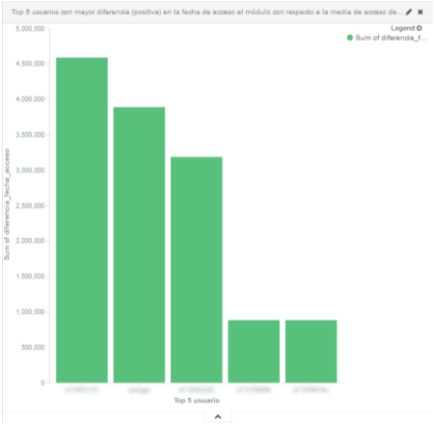


Figura 107. *Kibana*. *Dashboard* "ANÁLISIS DE UN MÓDULO" – Visualización: Top 5 usuarios con mayor diferencia (positiva) en la fecha de acceso al módulo con respecto a la media de acceso del curso

- Top 5 usuarios con mayor diferencia (positiva) en la fecha de acceso al módulo con respecto a la media de acceso del curso (Figura 107).
- Top 5 usuarios con mayor diferencia (negativa) en la fecha de acceso al módulo con respecto a la media de acceso del curso (Figura 108).

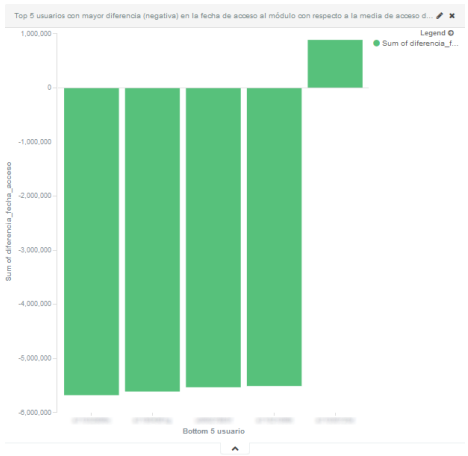


Figura 108. *Kibana*. *Dashboard* "ANÁLISIS DE UN MÓDULO" – Visualización: Top 5 usuarios con mayor diferencia (negativa) en la fecha de acceso al módulo con respecto a la media de acceso del curso

En el panel **"CALIFICACIONES FINALES"** se presenta la información sobre la evaluación final dada a los alumnos. Se puede ver una vista previa en la Figura 109. Se compone de los siguientes bloques.

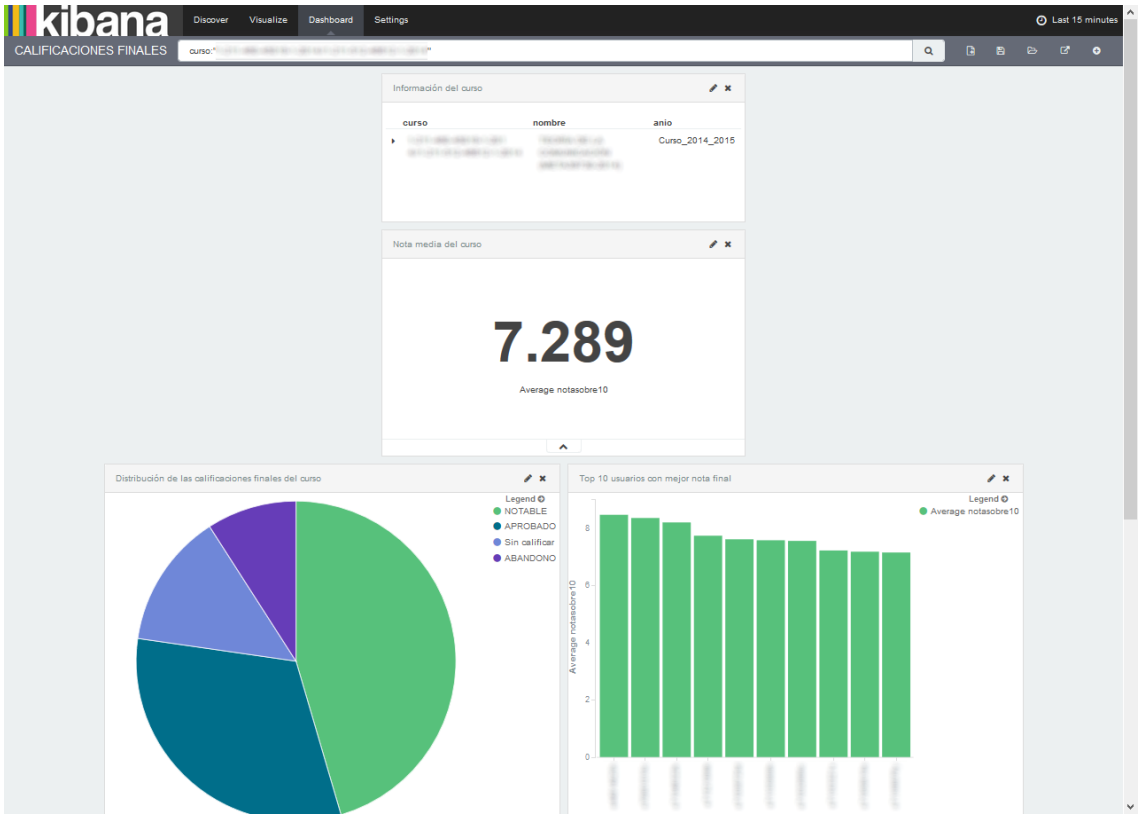


Figura 109. Vista previa del *dashboard* "CALIFICACIONES FINALES"

- Información del curso analizado (Figura 110).

Figura 110. *Kibana*. *Dashboard* "CALIFICACIONES" – Visualización: Información del curso

- Nota media del curso (Figura 111).

Figura 111. *Kibana*. *Dashboard* "CALIFICACIONES" – Visualización: Nota media del curso

- Distribución de las calificaciones finales del curso (Figura 112).

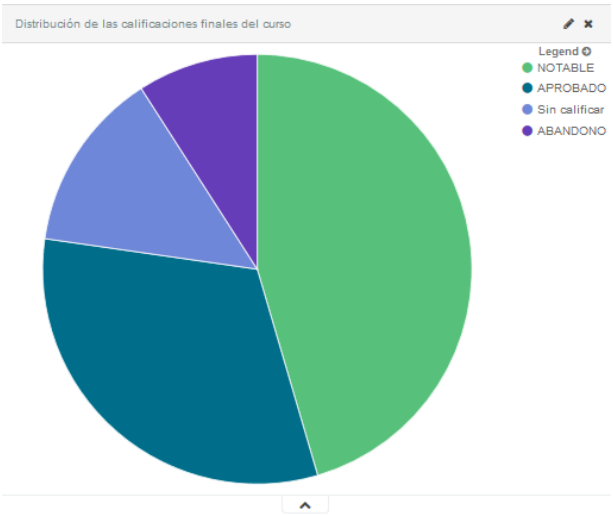


Figura 112. *Kibana*. *Dashboard* "CALIFICACIONES" – Visualización: Distribución de las calificaciones finales del curso

- Top 10 usuarios con mejor nota final (Figura 113).

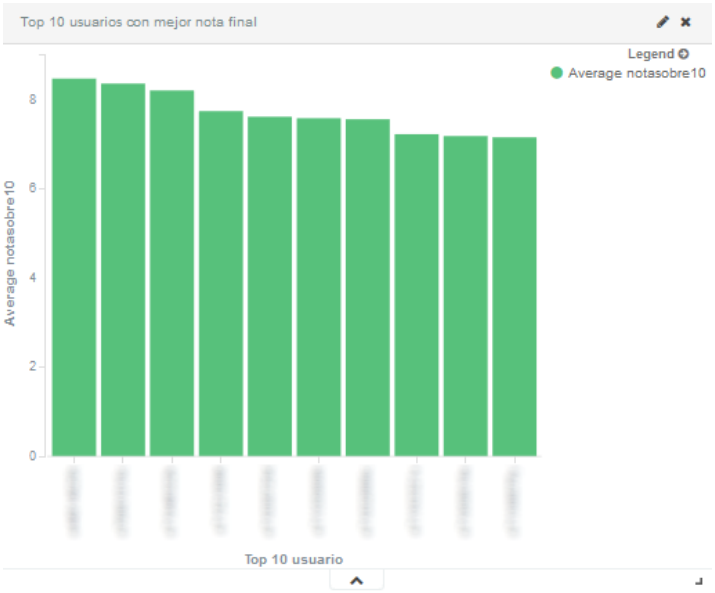


Figura 113. *Kibana*. *Dashboard* "CALIFICACIONES" – Visualización: Top 10 usuarios con mejor nota final

- Tabla con la información sobre las calificaciones finales del curso (Figura 114).

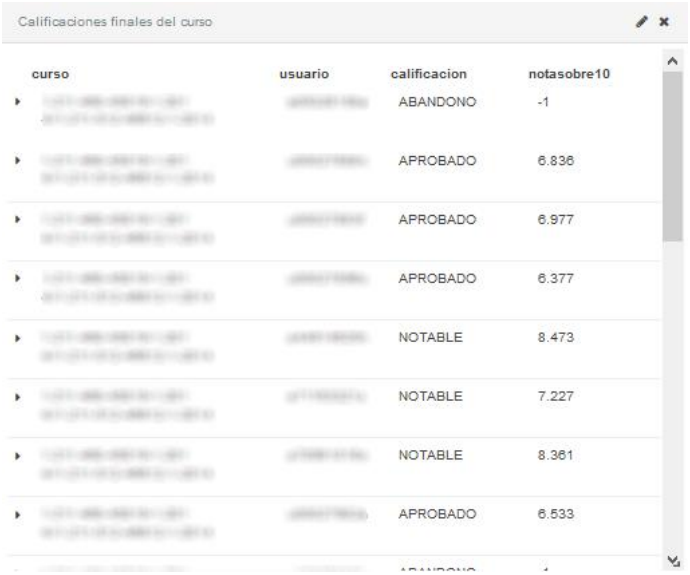


Figura 114. *Kibana*. *Dashboard* "CALIFICACIONES" – Visualización: calificaciones finales del curso

En el panel **"CLASIFICACIÓN"** se representan los resultados de la aplicación del algoritmo de clasificación sobre los alumnos del curso con el objetivo de predecir su nota. En la Figura 115 se muestra la vista previa de este panel, que estará compuesto de los gráficos que se enumeran a continuación.

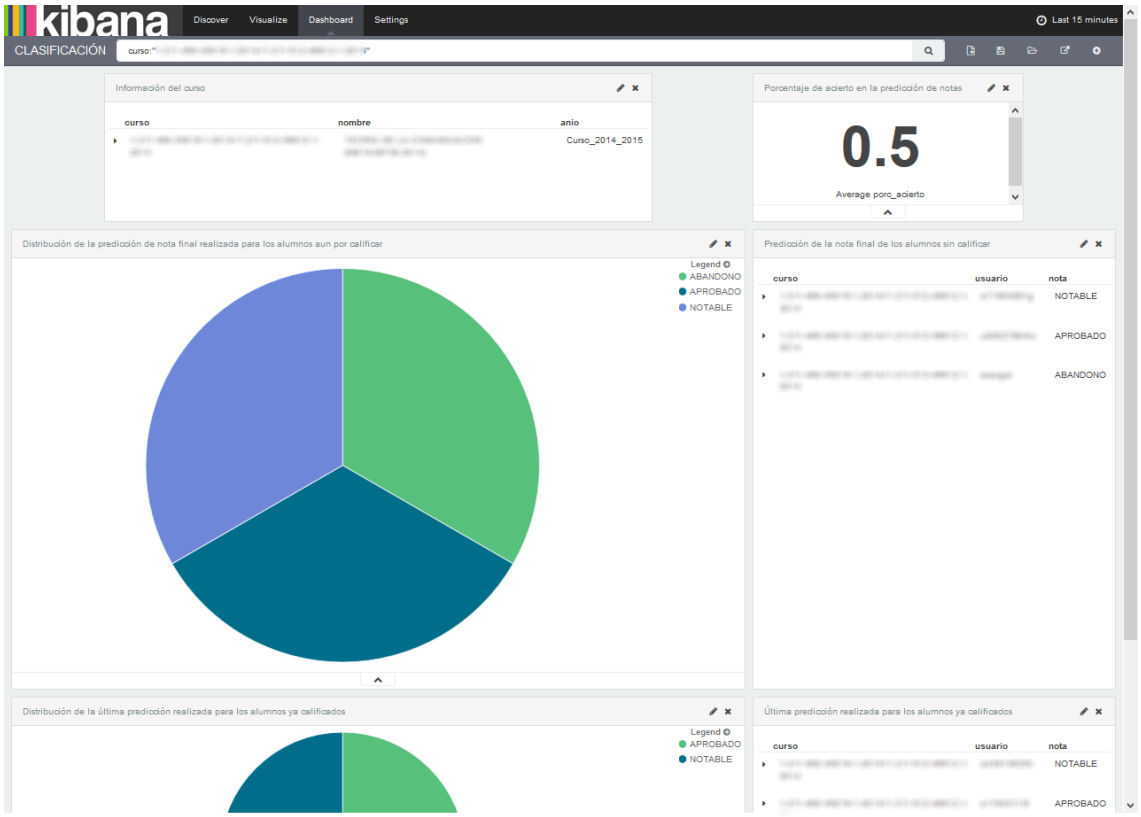


Figura 115. Vista previa del *dashboard* "CLASIFICACIÓN"

- Información del curso, similar al mostrado en la Figura 110 .
- Porcentaje de acierto en la predicción de notas, en tanto por uno (Figura 116).

Figura 116. *Kibana*. *Dashboard* "CLASIFICACIÓN" – Visualización: Porcentaje de acierto en la predicción de notas

- Distribución de la predicción de la nota final realizada para los alumnos aun por calificar (Figura 117).

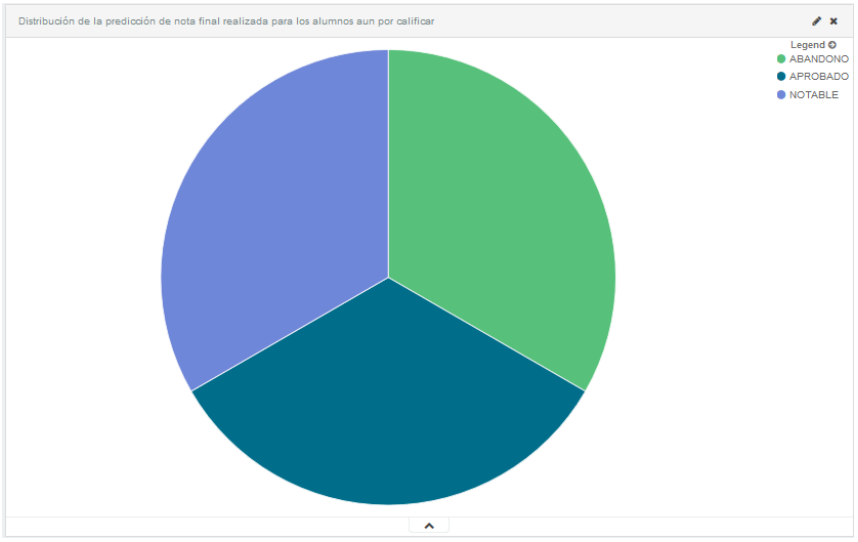


Figura 117. *Kibana*. *Dashboard* "CLASIFICACIÓN" – Visualización: Distribución de la predicción de nota final realizada para los alumnos aun por calificar

- Predicción de la nota final de los alumnos sin calificar (Figura 118).

Figura 118. *Kibana*. *Dashboard* "CLASIFICACIÓN" – Visualización: Predicción de la nota final de los alumnos sin calificar

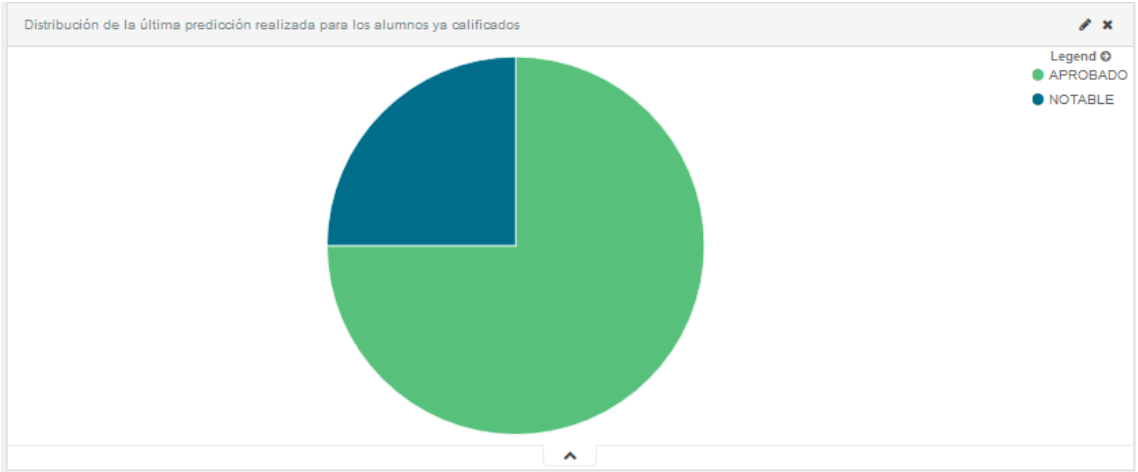


Figura 119. *Kibana*. *Dashboard* "CLASIFICACIÓN" – Visualización: Distribución de la última predicción realizada para los alumnos ya calificados

- Distribución de la última predicción realizada para los alumnos ya calificados (Figura 119).

- Última predicción realizada para los alumnos ya calificados (Figura 120).

Figura 120. *Kibana*. *Dashboard* "CLASIFICACIÓN" – Visualización: Última predicción realizada para los alumnos ya calificados

En el panel **"*CLUSTERING*"** se representan los resultados de la aplicación del algoritmo de *clustering* sobre los usuarios del curso. Su vista previa se puede observar en la Figura 121. Se compone de los siguientes bloques.

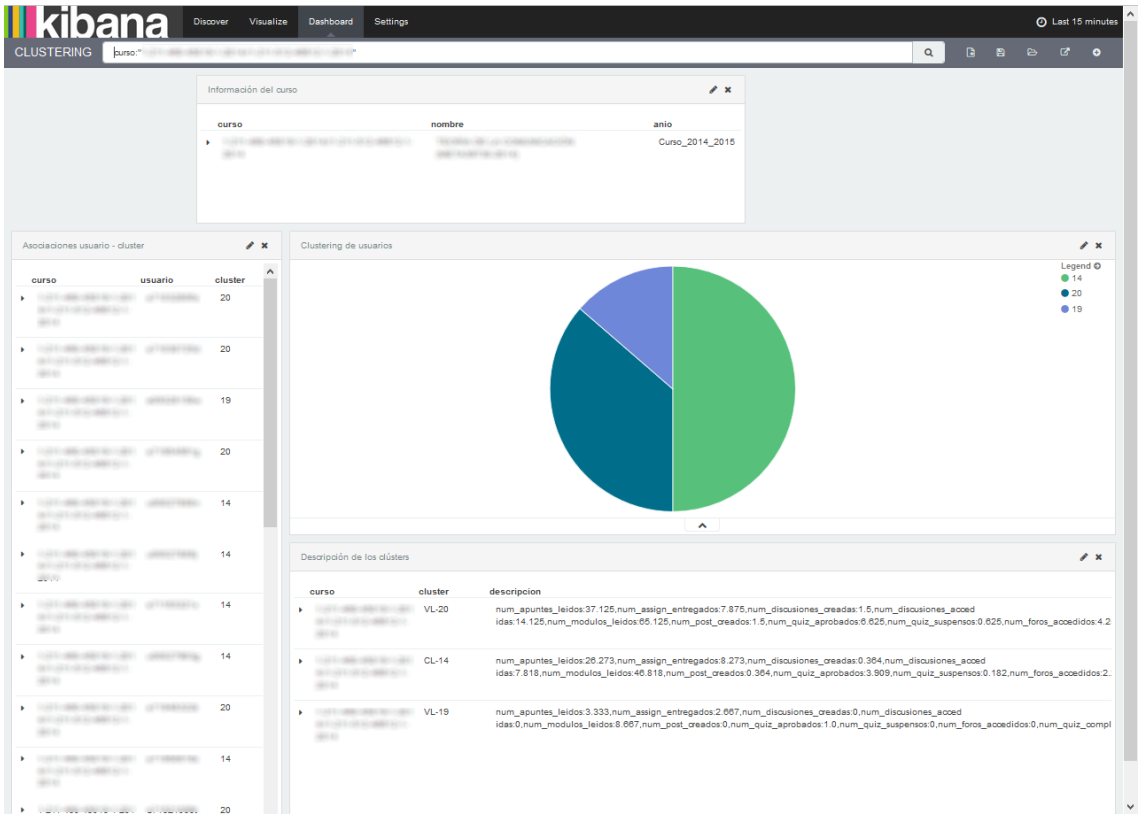


Figura 121. Vista previa del *dashboard* "*CLUSTERING*"

- Información del curso, similar al mostrado en la Figura 110.
- Asociaciones usuario - *cluster* obtenidas en la aplicación del algoritmo (Figura 122).

Figura 122. *Kibana*. *Dashboard* "CLUSTERING" – Visualización: Asociaciones usuario - *cluster*

- Representación del *clustering* de usuarios obtenido (Figura 123).

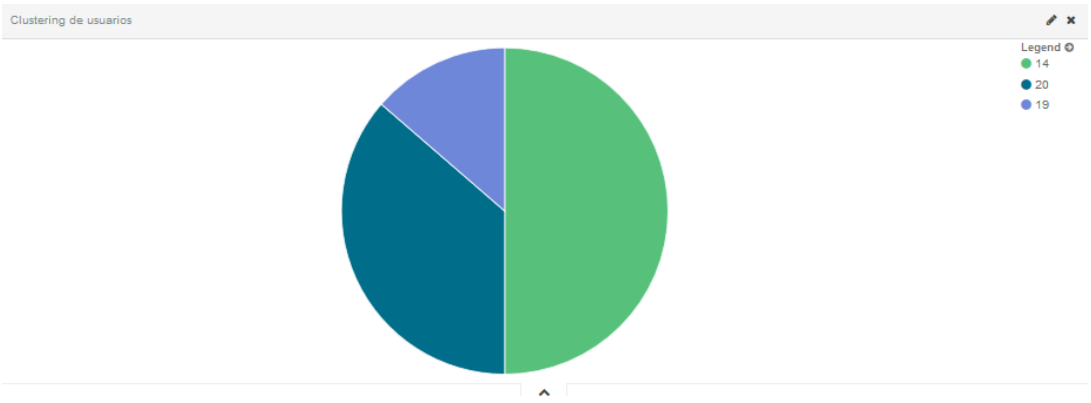


Figura 123. *Kibana*. *Dashboard* "*CLUSTERING*" – Visualización: *Clustering* de usuarios

- Descripción de los *clusters* (Figura 124).

Figura 124. *Kibana*. *Dashboard* "*CLUSTERING*" – Visualización: Descripción de los *clusters*

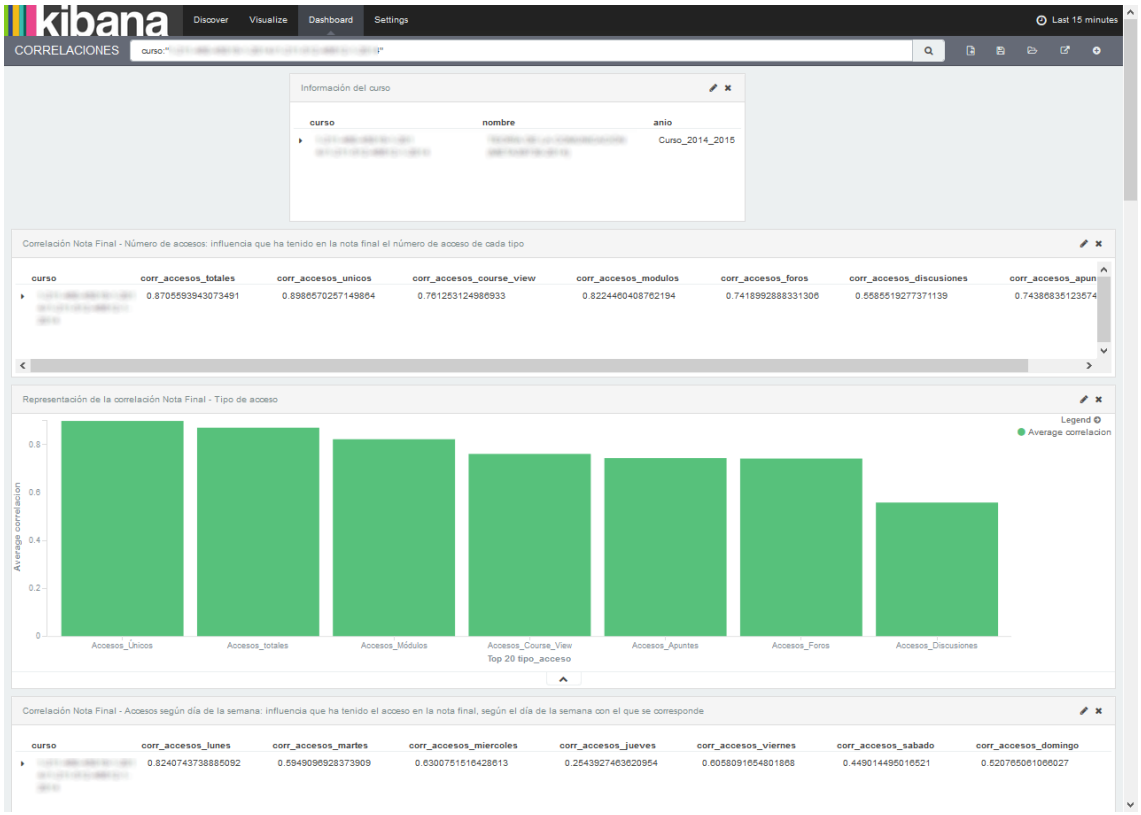


Figura 125. Vista previa del *dashboard* "CORRELACIONES"

En el apartado **"CORRELACIONES"** se muestra la información sobre la influencia entre la actividad de los alumnos en *Moodle* y sus notas finales, organizada según varios criterios enumerados a continuación. Su vista previa se puede observar en la Figura 125.

- Información del curso, similar al mostrado en la Figura 110.
- Correlación entre la nota final y el número de accesos, es decir, influencia que ha tenido en la nota final el número de accesos de cada tipo (Figura 126).

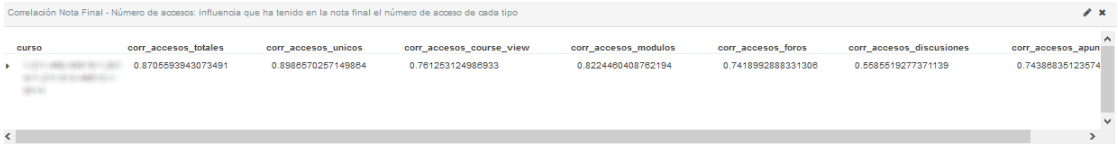


Figura 126. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Correlación Nota Final - Tipo de acceso

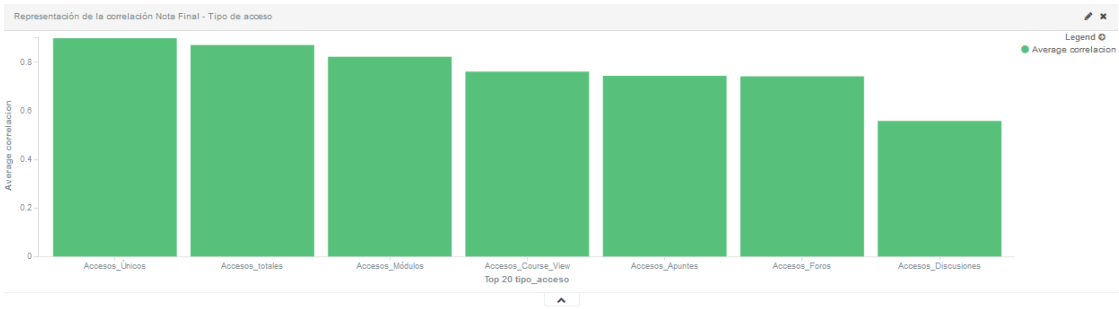


Figura 127. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Representación correlación Nota Final – Tipo de acceso

- Representación gráfica de la relación anterior (Figura 127).
- Correlación entre la nota final y el número de accesos según día de la semana, es decir, influencia que ha tenido el acceso en la nota final, según el día de la semana con el que se corresponde (Figura 128).

Figura 128. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Correlación Nota Final – Accesos según día de la semana

Figura 129. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Representación correlación Nota Final – Accesos según día de la semana

- Representación gráfica de la relación anterior (Figura 129).
- Correlación entre la nota final y la actividad realizada, es decir, influencia de la realización de actividad (*assign* y *quiz*) en la nota final (Figura 130).

Figura 130. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Correlación Nota Final – Actividades

- Representación gráfica de la relación anterior (Figura 131).

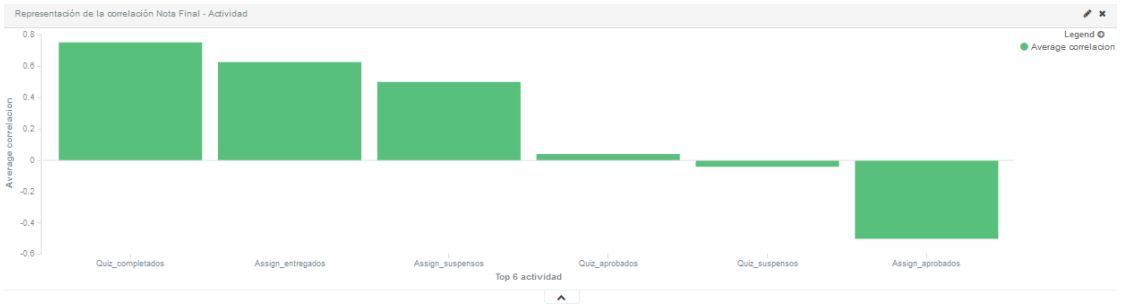


Figura 131. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Representación correlación Nota Final – Actividades

- Correlación entre la nota final y el origen de los accesos, es decir, influencia en la nota final de los accesos en función de si se realizaron desde dentro o fuera de la Universidad (Figura 132).

Figura 132. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Correlación Nota Final – Origen de los accesos

Figura 133. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Representación correlación Nota Final – Origen de los accesos

- Representación gráfica de la relación anterior (Figura 133).
- Correlación entre la nota final y el número de acceso a módulos, es decir, influencia del número de accesos a cada módulo sobre la nota final obtenida (Figura 134).

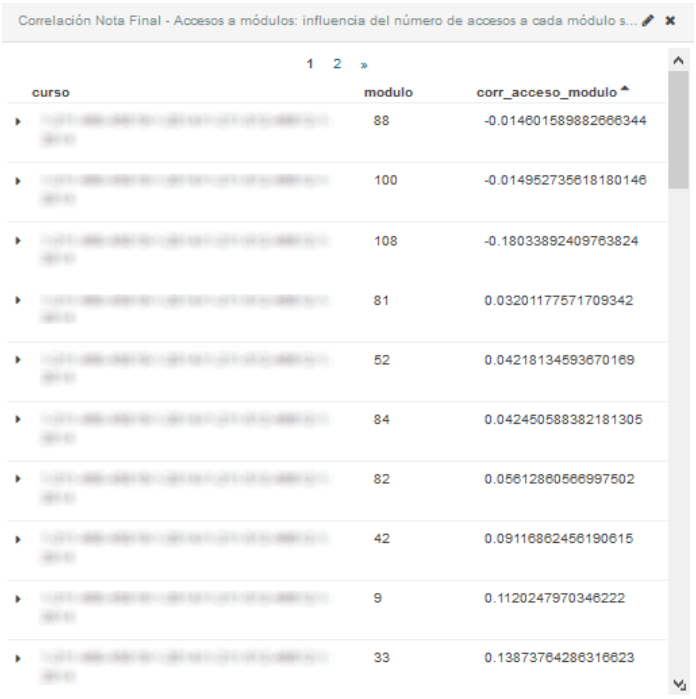


Figura 134. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Correlación Nota Final – Accesos a módulos

- Representación gráfica de la relación anterior (Figura 135).

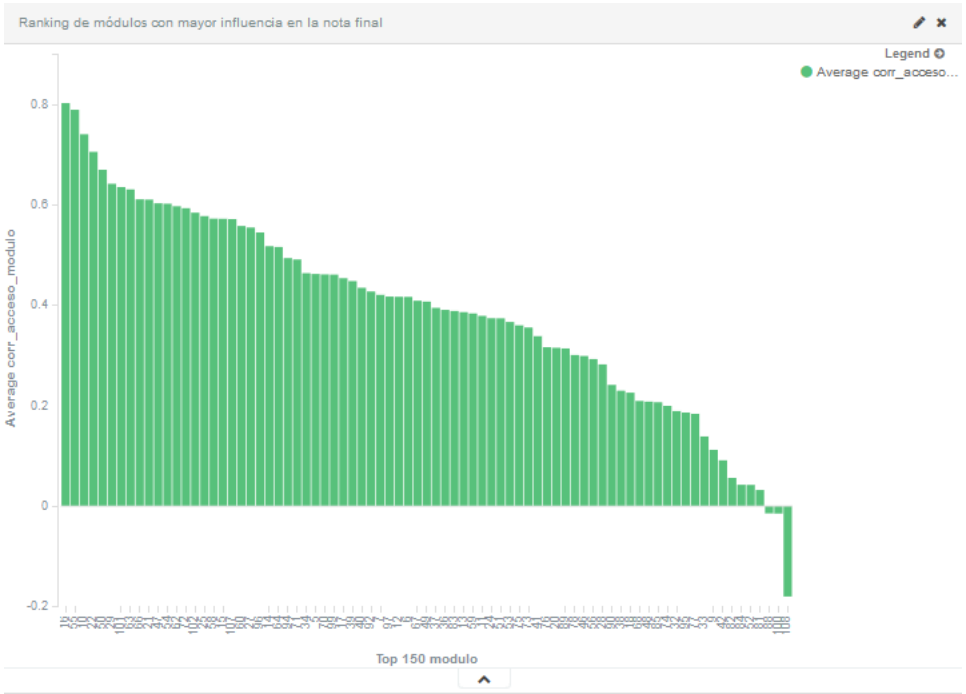


Figura 135. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Representación correlación Nota Final – Accesos a módulos

- Correlación entre la nota final y el uso de módulos, es decir, influencia del uso de los módulos más representativos sobre la nota final (Figura 136).

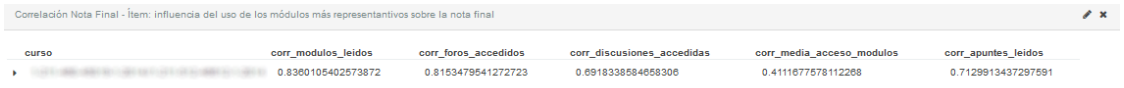


Figura 136. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Correlación Nota Final – Ítem

- Representación gráfica de la relación anterior (Figura 137).

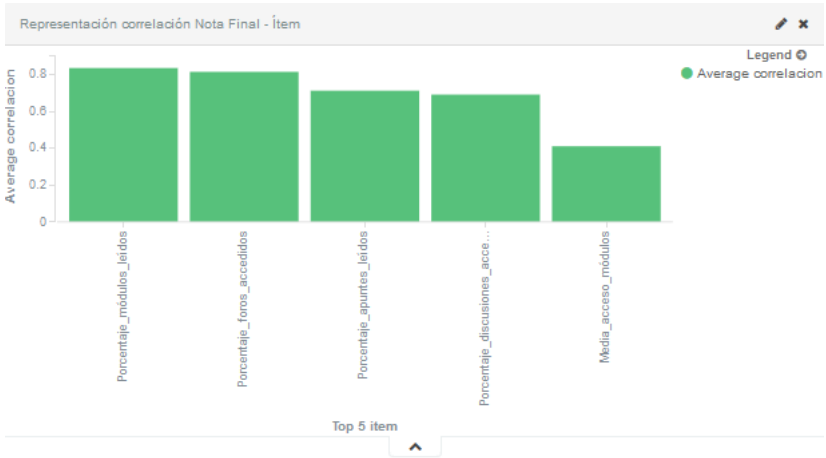


Figura 137. *Kibana*. *Dashboard* "CORRELACIONES" – Visualización: Representación correlación Nota Final – Ítem

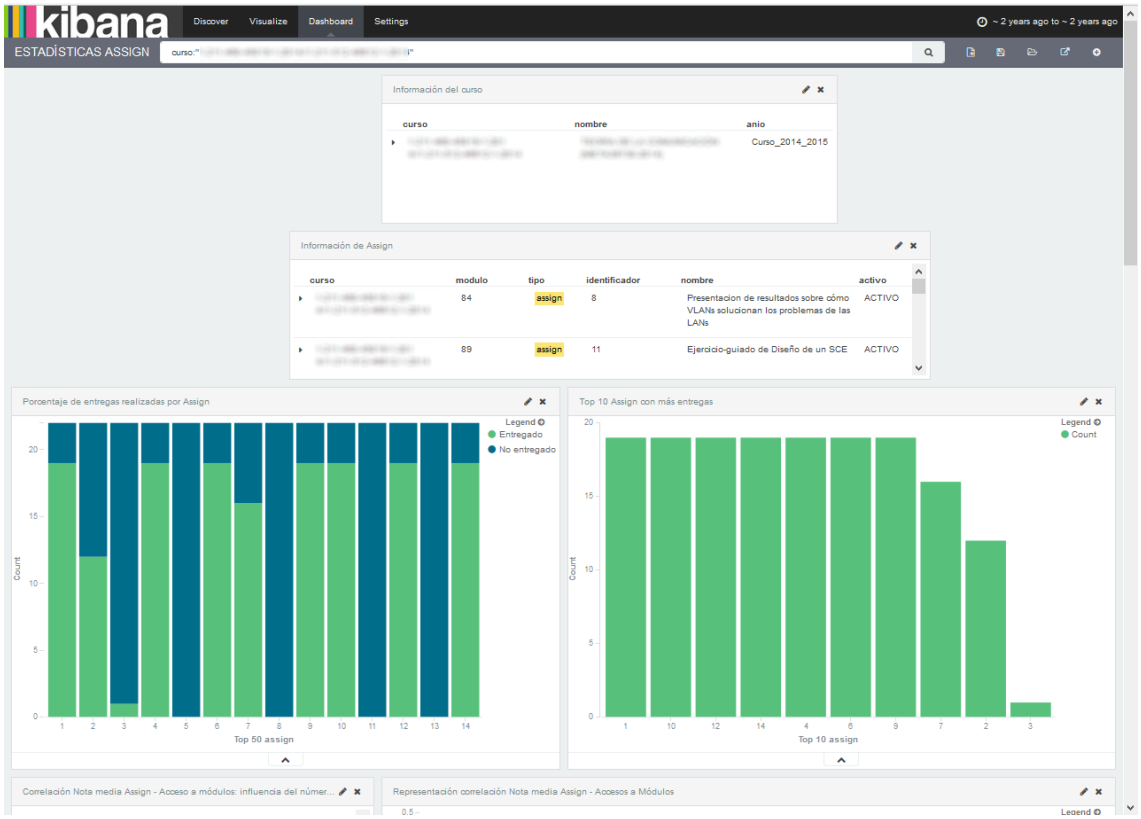


Figura 138. Vista previa del *dashboard* "ESTADÍSTICAS ASSIGN"

En el panel **"ESTADÍSTICAS *ASSIGN*"** se representan las estadísticas genéricas de los *assign* de un curso. Se puede ver su vista previa en la Figura 138, la cual está compuesta de las visualizaciones que se enumeran a continuación.

- Información del curso, similar al mostrado en la Figura 110.
- Información de los *Assign* publicados en el curso (Figura 139).

Figura 139. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN*" – Visualización: Información de *Assign*

- Porcentaje de entregas realizadas por *Assign* (Figura 140).

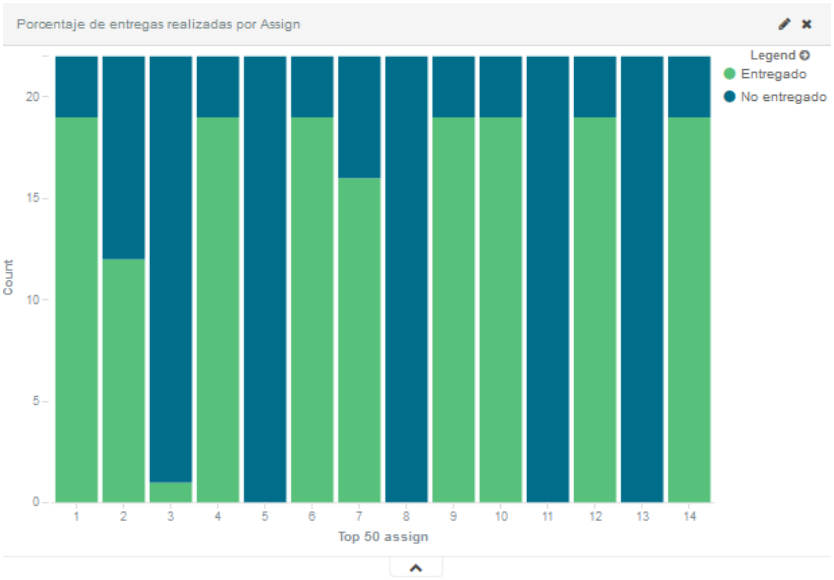


Figura 140. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN*" – Visualización: Porcentaje de entregas realizadas por *Assign*

- Top 10 *Assign* con más entregas (Figura 141).

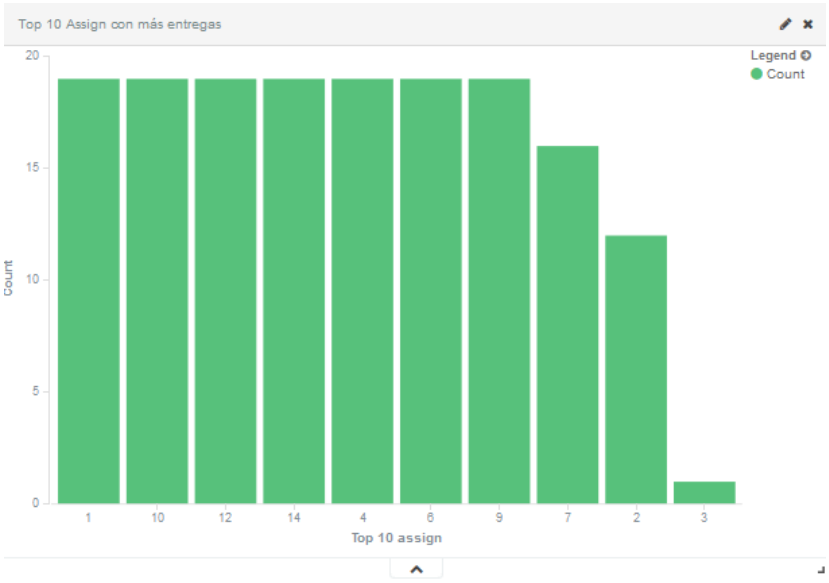


Figura 141. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN*" – Visualización: Top 10 *Assign* con más *entregas*

- Correlación entre la nota media obtenida en cada *assign* y el acceso a módulos, es decir, influencia del número de acceso a cada módulo en la nota media de todos los alumnos en cada *assign* del curso (Figura 142).

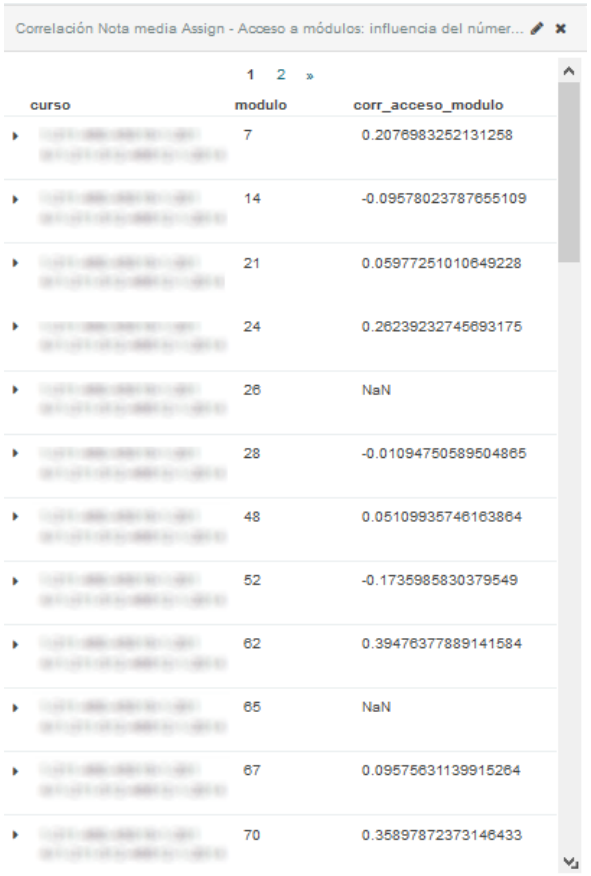


Figura 142. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN*" – Visualización: Correlación Nota media *Assign* – Accesos a módulos

- Representación gráfica de la relación anterior (Figura 143).

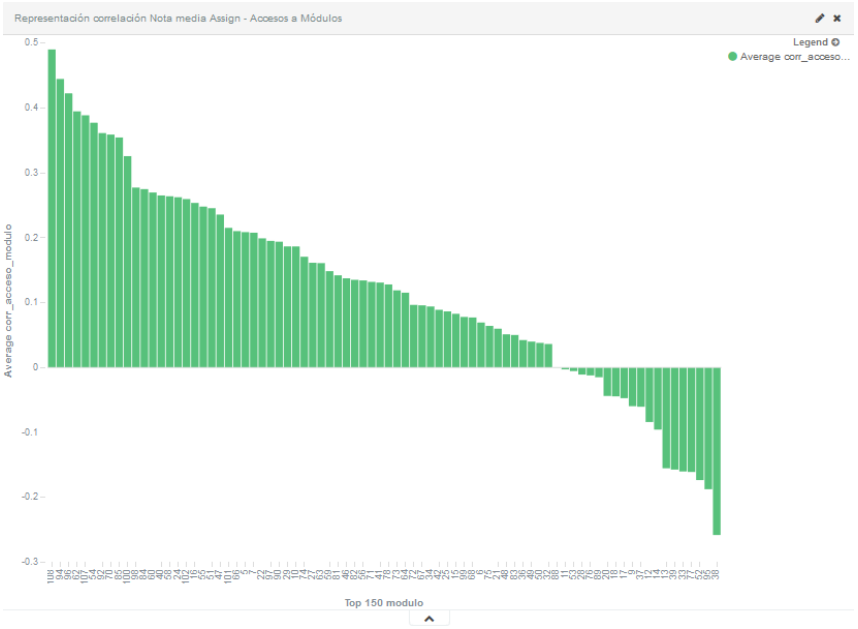


Figura 143. *Kibana*. *Dashboard* "ESTADÍSTICAS ASSIGN" – Visualización: Representación correlación Nota media *Assign* – Accesos a módulos

- Porcentaje de relaciones *assign* – alumno calificados en el curso (Figura 144).

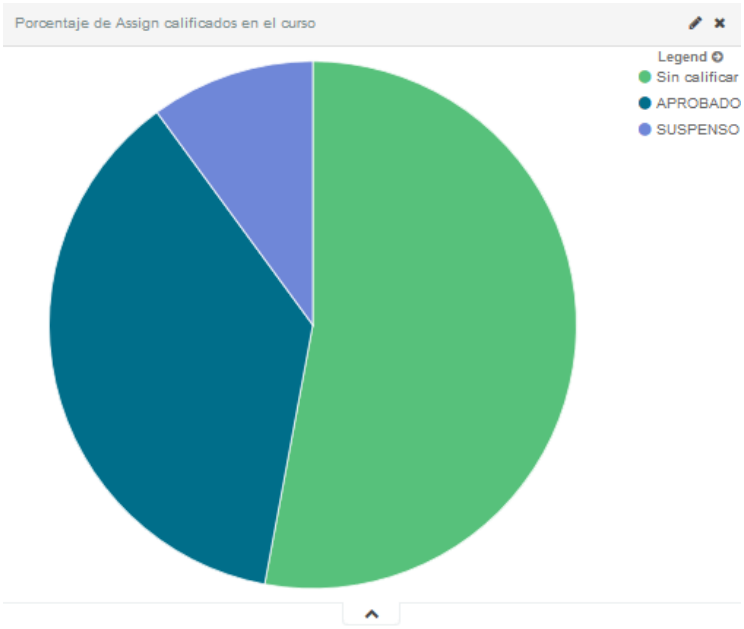


Figura 144. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN*" – Visualización: Porcentaje de *Assign* calificados en el curso

- Calificaciones realizadas por *assign* (Figura 145).

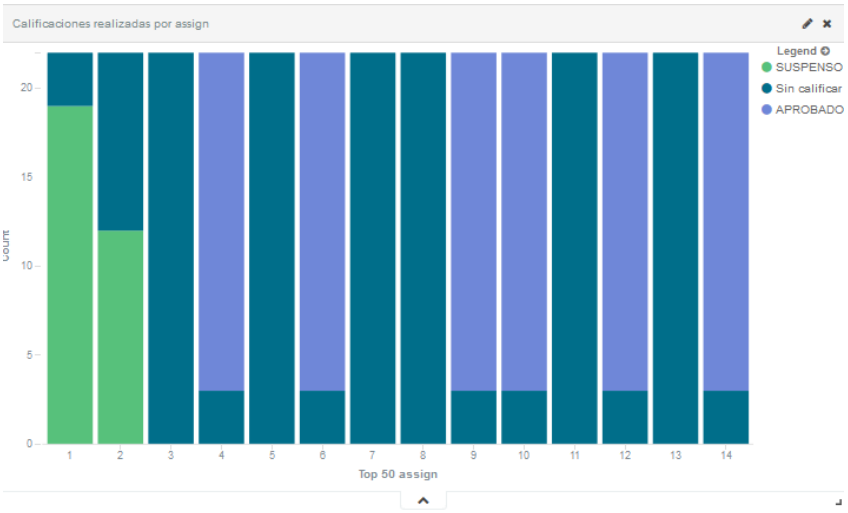


Figura 145. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN*" – Visualización: Calificaciones realizadas por *Assign*

- Top 10 *assign* con mayor nota media (Figura 146).

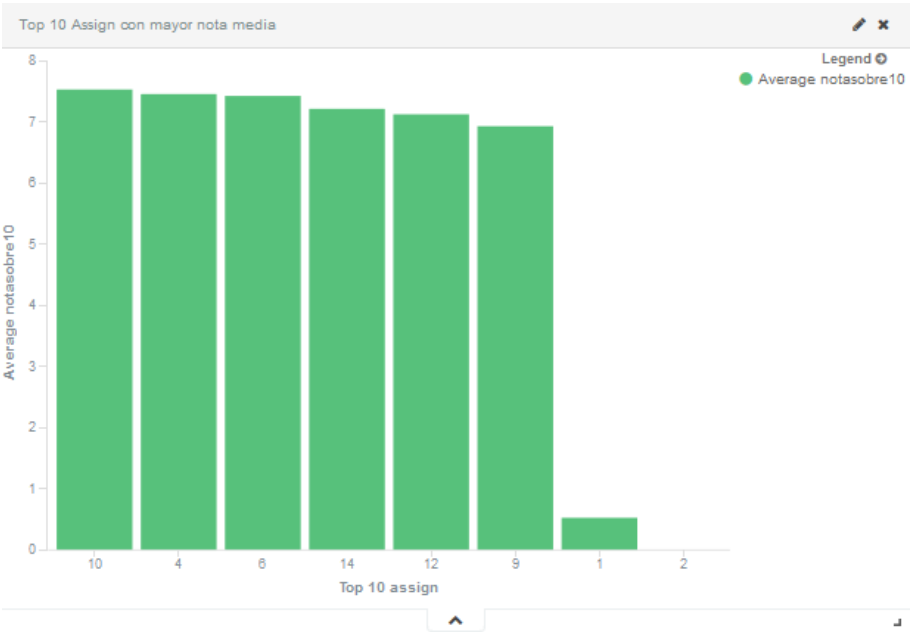


Figura 146. *Kibana*. *Dashboard* "ESTADÍSTICAS ASSIGN" – Visualización: Top 10 *Assign* con mayor nota media

- Top 10 alumnos con mejor nota media en *assign* (Figura 147).

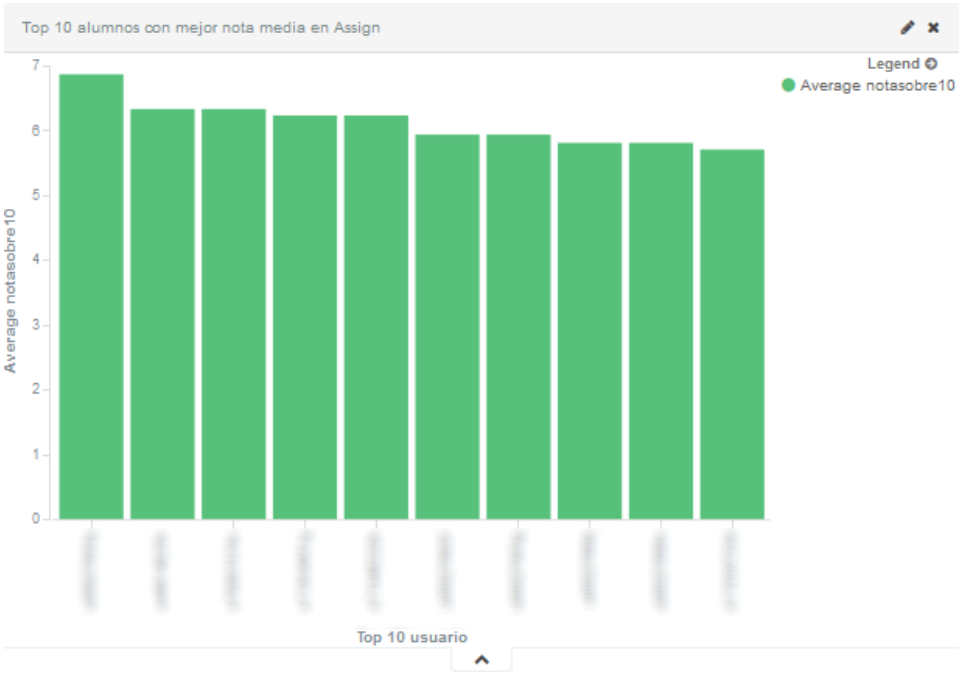


Figura 147. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN*" – Visualización: Top 10 alumnos con mejor nota media en *assign*

- Distribución de las calificaciones en *Assign* realizadas por alumno (Figura 148).

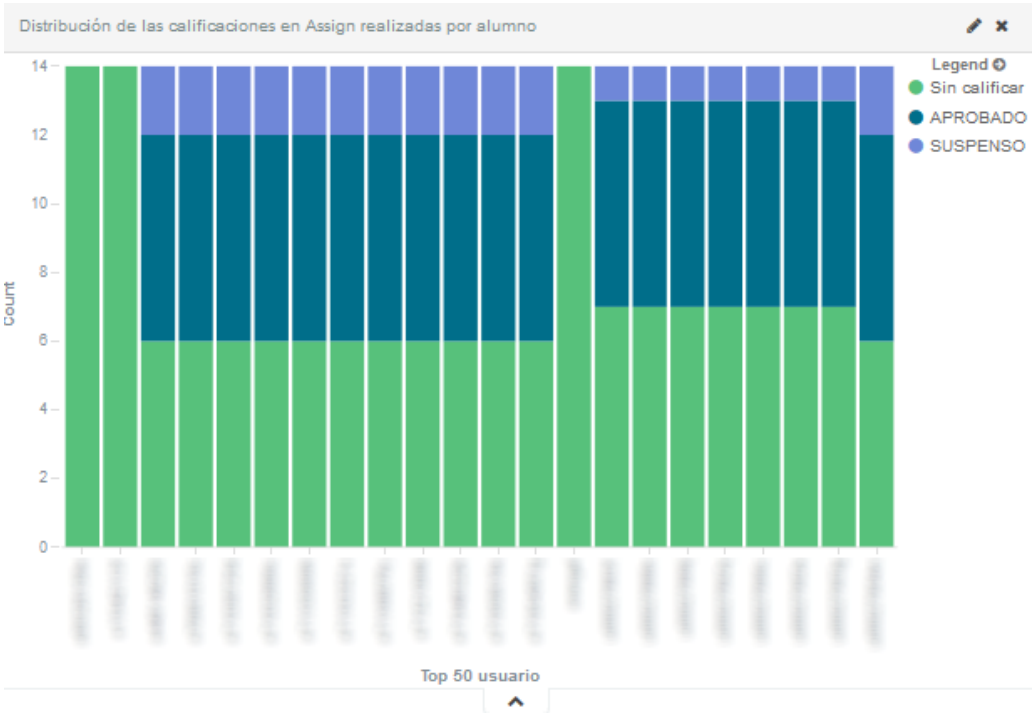


Figura 148. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN*" – Visualización: Distribución de las calificaciones en *Assign* realizadas por alumno

- Porcentaje de entregas de *assign* por alumno (Figura 149).

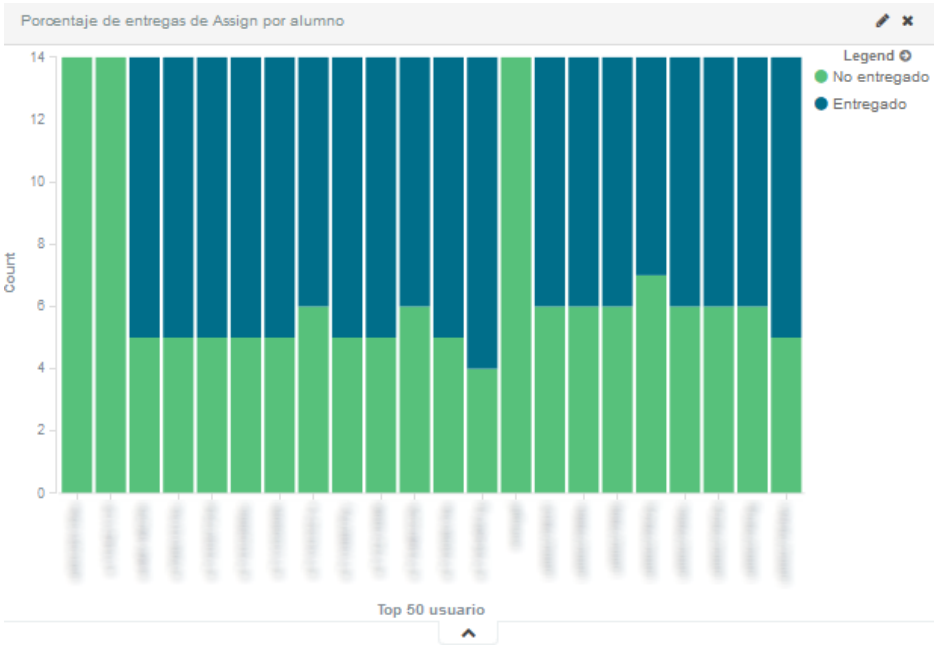


Figura 149. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN*" – Visualización: Porcentaje de entregas de *Assign* por alumno

En el panel **"ESTADÍSTICAS *ASSIGN* – USUARIO"** se representan las estadísticas de actividad de un alumno, sobre los *assign* de un curso. Se puede observar su vista previa en la Figura 150. Se compone de los siguientes bloques.

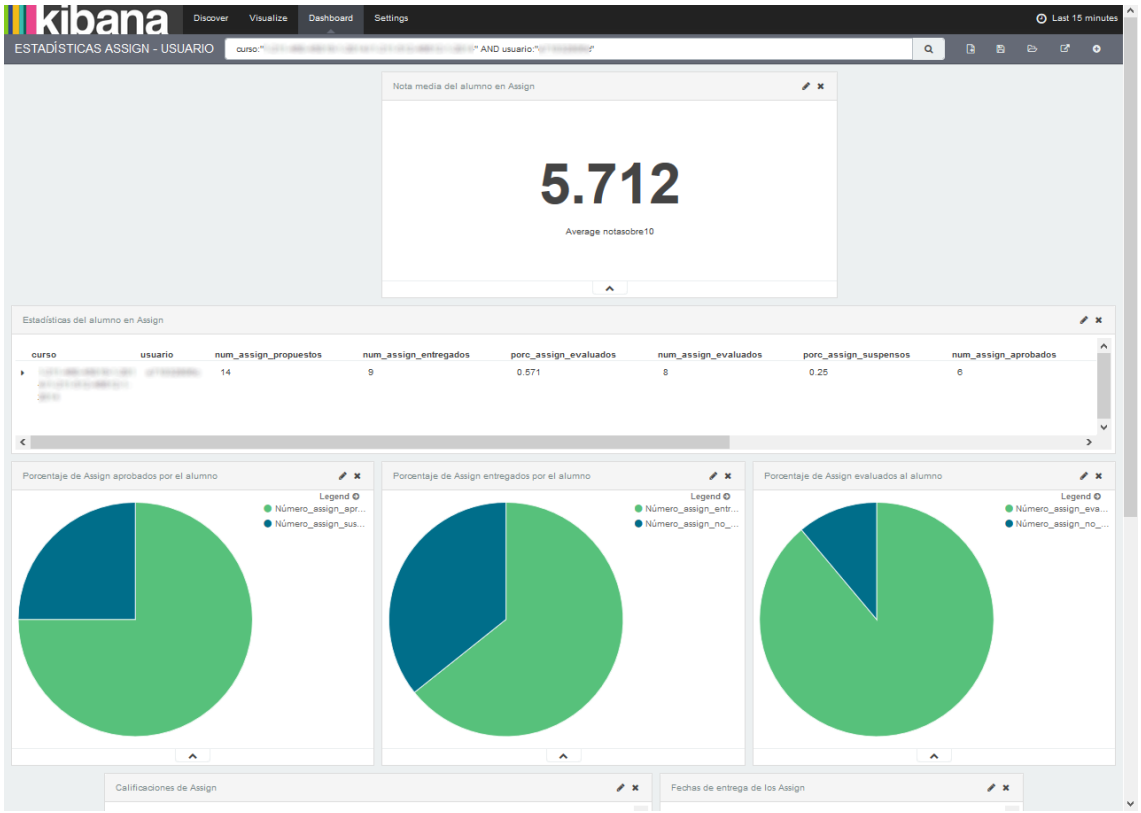


Figura 150. Vista previa del *dashboard* "ESTADÍSTICAS *ASSIGN* – USUARIO"

- Nota media del alumno en los *assign* del curso (Figura 151).

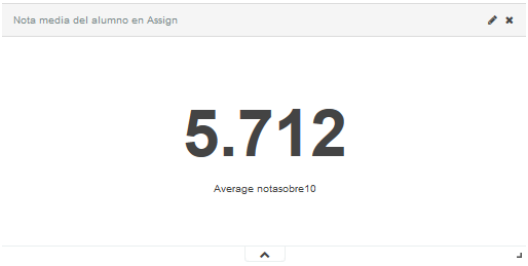


Figura 151. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN* – USUARIO" – Visualización: Nota media del alumno en *Assign*

- Estadísticas del alumno sobre la actividad en *assign* (Figura 152).

Figura 152. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN* – USUARIO" – Visualización: Estadísticas del alumno en *Assign*

- Porcentaje de *assign* aprobados por el alumno (Figura 153).

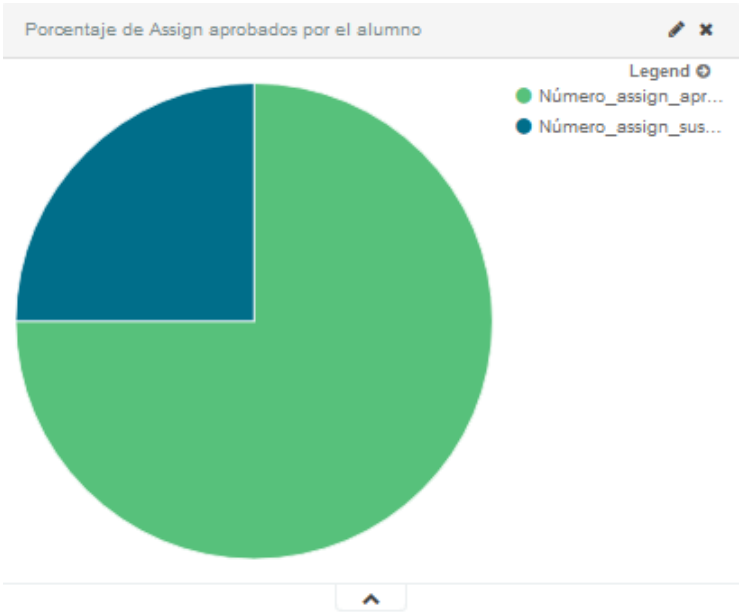


Figura 153. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN* – USUARIO" – Visualización: Porcentaje de *Assign* aprobados por el alumno

- Porcentaje de *assign* entregados por el alumno (Figura 154).

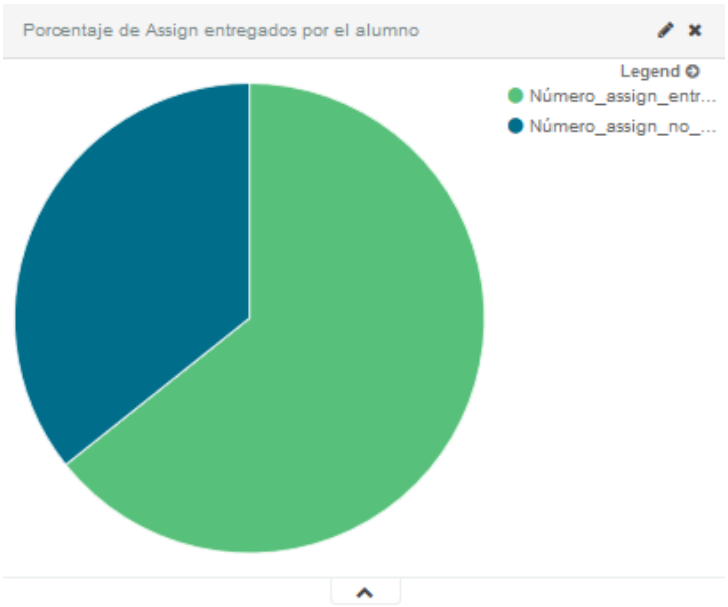


Figura 154. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN* – USUARIO" – Visualización: Porcentaje de *Assign* entregados por el alumnos

- Porcentaje de *assign* evaluados al alumno (Figura 155).

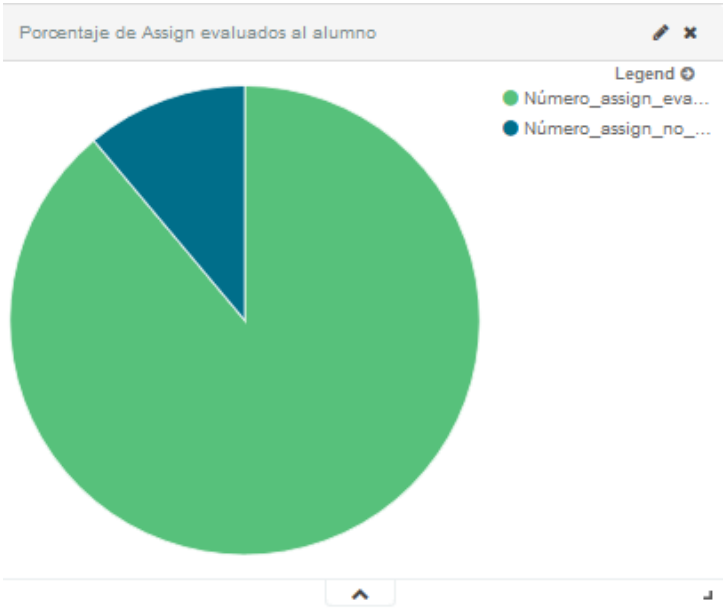


Figura 155. *Kibana*. *Dashboard* "ESTADÍSTICAS ASSIGN – USUARIO" – Visualización: Porcentaje de *Assign* evaluados al alumno

- Tabla de calificaciones del alumno en *assign* (Figura 156).

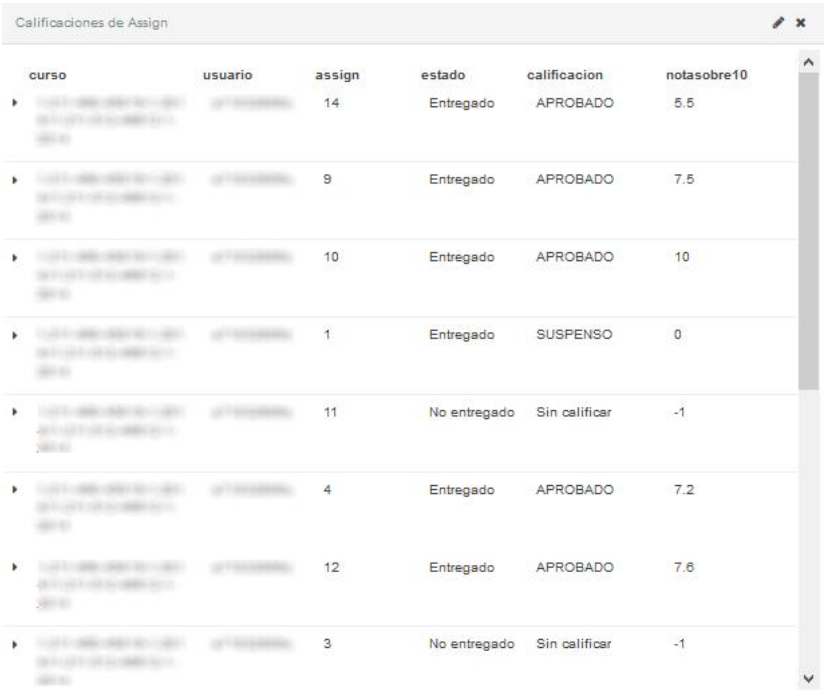


Figura 156. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN* – USUARIO" – Visualización: Calificaciones de *Assign*

- Fechas de entregas de los *assign* por el alumno (Figura 157).

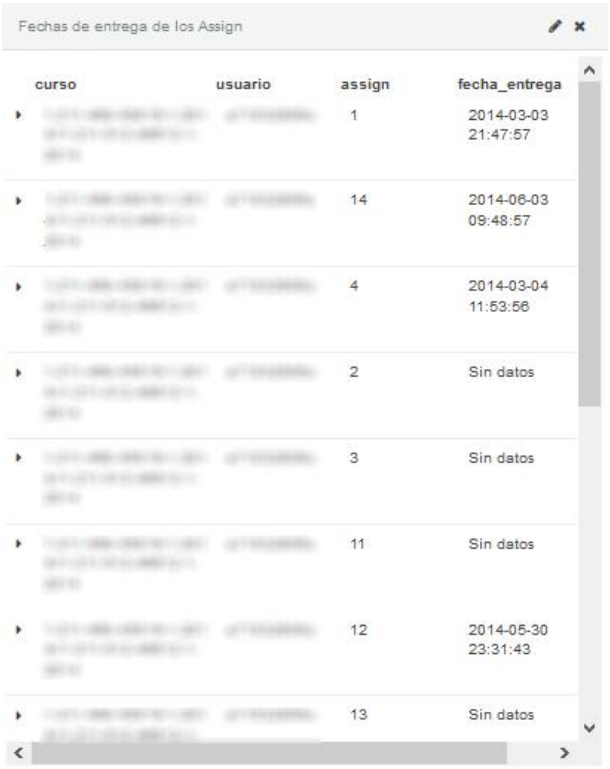


Figura 157. *Kibana*. *Dashboard* "ESTADÍSTICAS *ASSIGN* – USUARIO" – Visualización: Fechas de entrega de los *Assign*

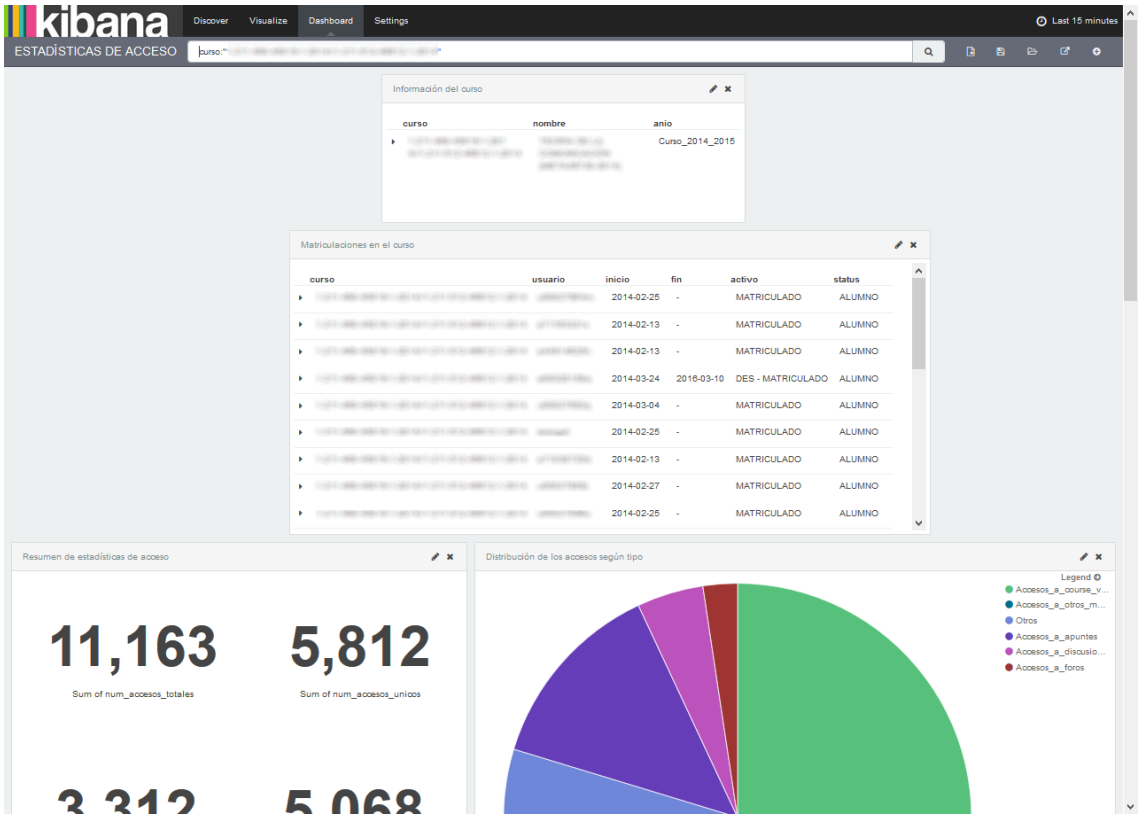


Figura 158. Vista previa del dashboard "ESTADÍSTICAS DE ACCESO"

En el panel **"ESTADÍSTICAS DE ACCESO"** se representan las estadísticas de acceso a los elementos del curso, como se puede observar en la vista previa de la Figura 158. Se compone de las siguientes visualizaciones.

- Información del curso, similar al mostrado en la Figura 110.
- Tabla de usuarios matriculados en el curso (Figura 159).

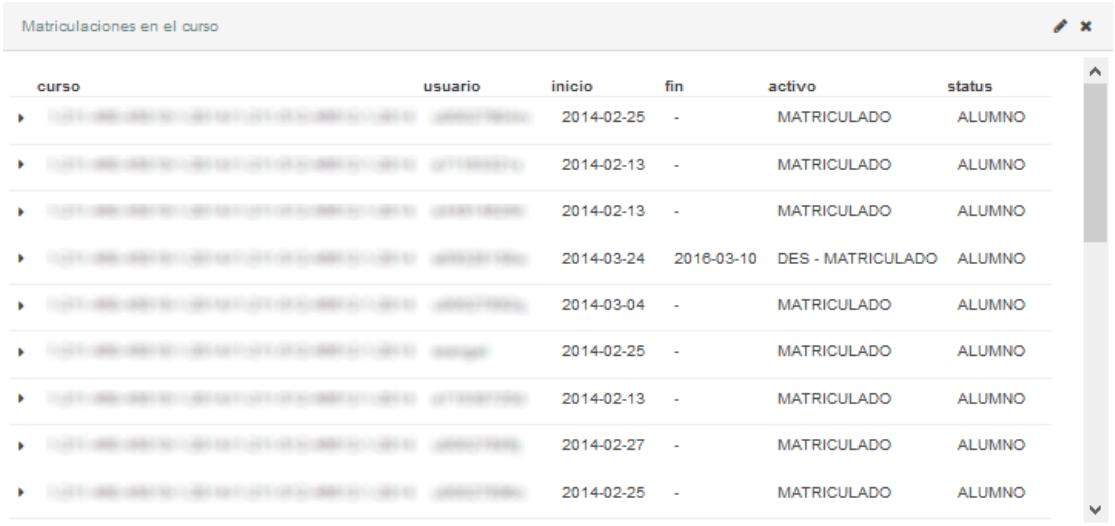


Figura 159. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO" – Visualización: Matriculaciones en el curso

- Tabla resumen de las estadísticas de acceso al curso (Figura 160).

Figura 160. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO" – Visualización: Resumen estadísticas de acceso

- Distribución de los accesos (considerados en el análisis) según tipo (Figura 161).

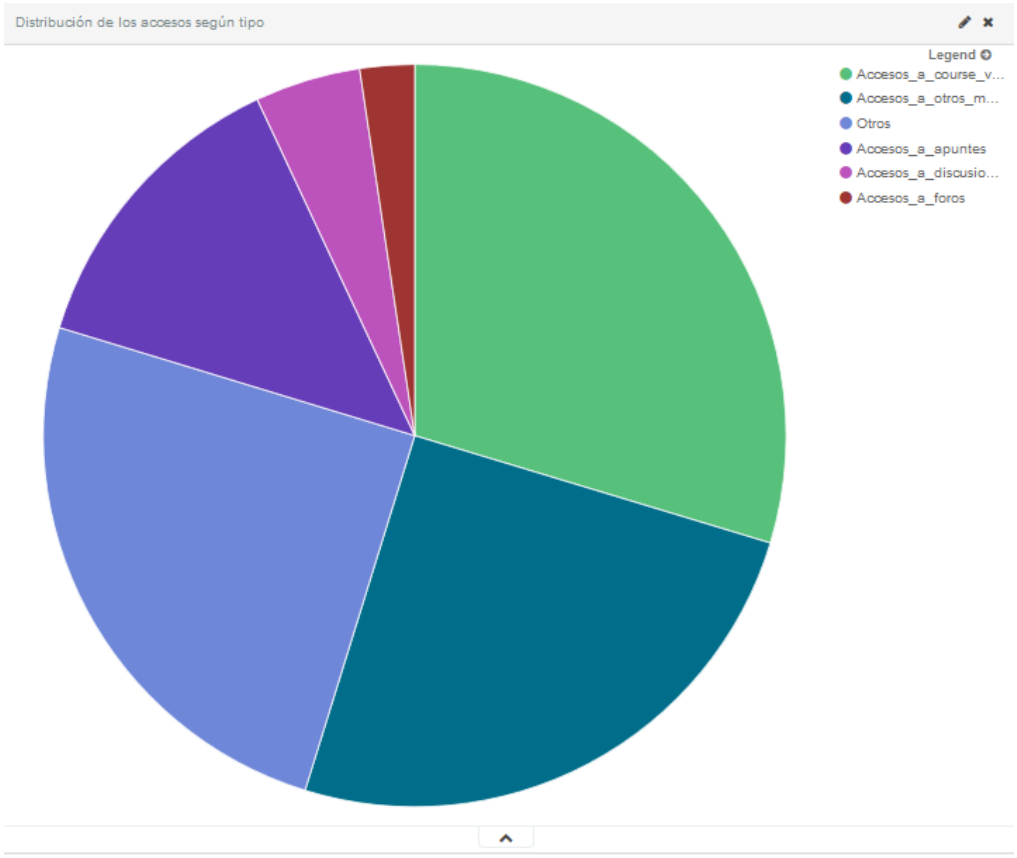


Figura 161. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO" – Visualización: Distribución de los accesos según tipo

- Distribución de los accesos según día de la semana (Figura 162 y Figura 163).

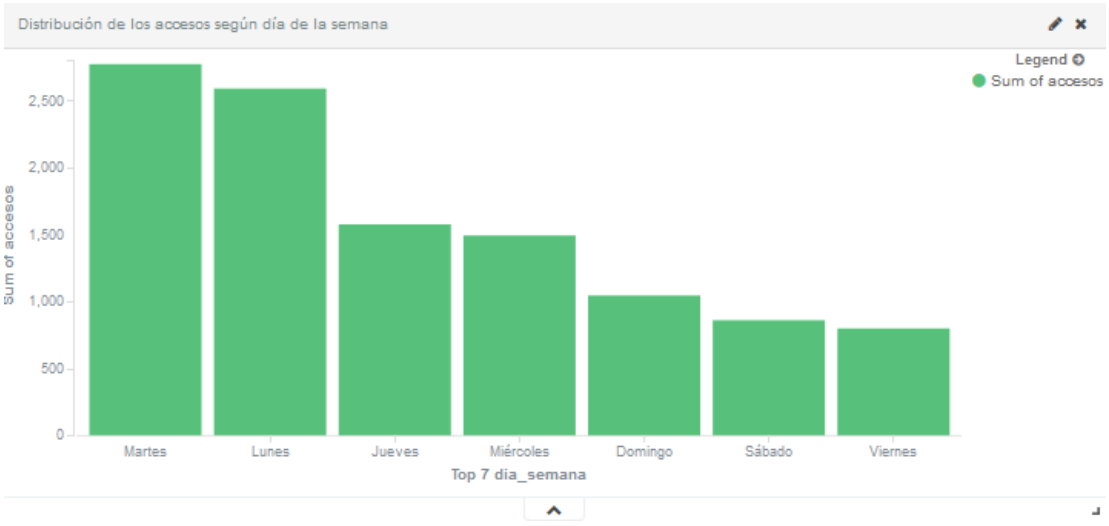


Figura 162. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO" – Visualización: Distribución de los accesos según día de la semana I

Figura 163. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO" – Visualización: Distribución de los accesos según día de la semana II

- Distribución de los accesos según origen (Figura 164).

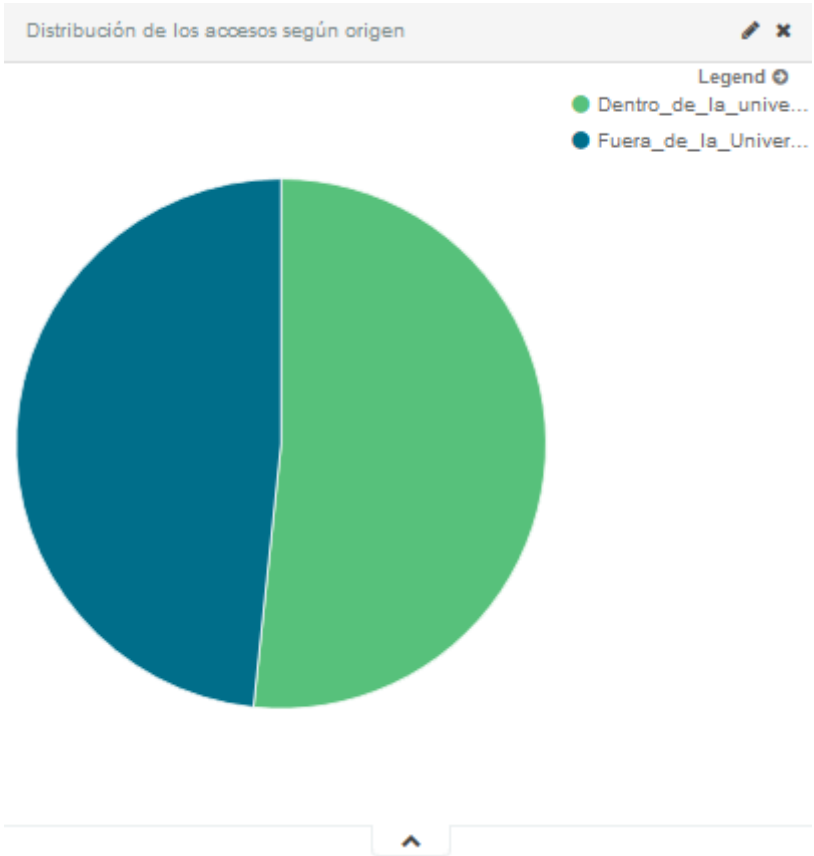


Figura 164. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO" – Visualización: Distribución de los accesos según origen

- Top 10 usuarios con más accesos totales (Figura 165). De forma análoga se podrían analizar otros tipos de accesos.

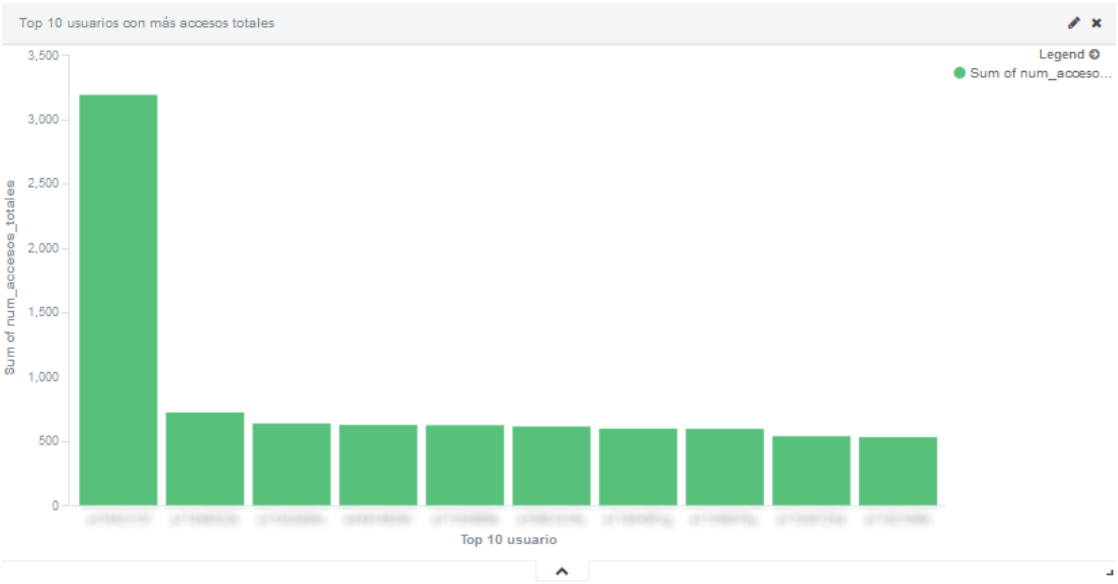


Figura 165. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO" – Visualización: Top 10 usuarios con más accesos totales

- Top 10 usuarios con más accesos, distribuidos por día de la semana (Figura 166).

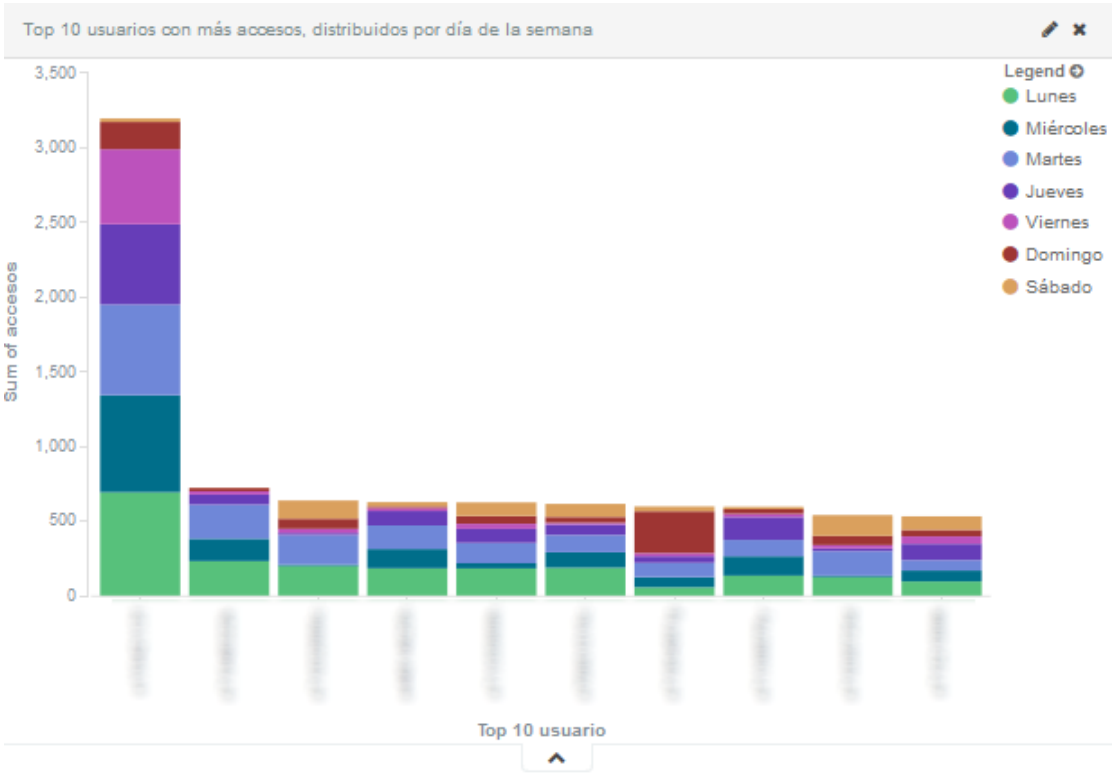


Figura 166. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO" – Visualización: Top 10 usuarios con más accesos, distribuidos por día de la semana

- Top 10 usuarios con más accesos, distribuidos por tipo de acceso (Figura 167).

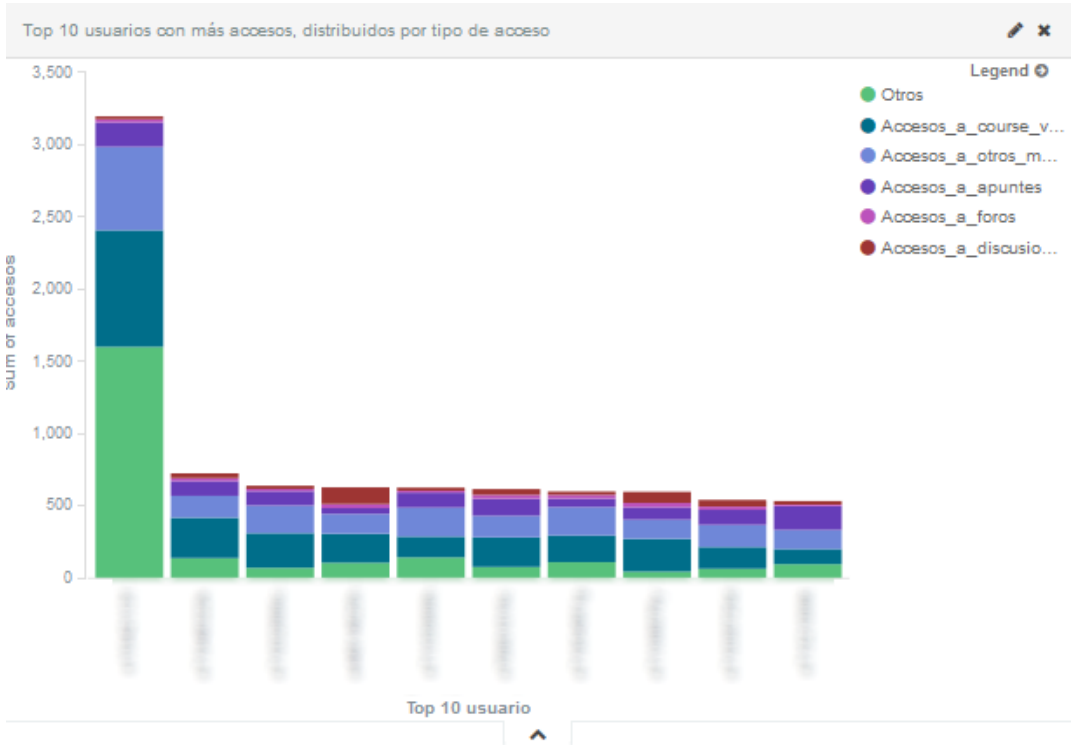


Figura 167. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO" – Visualización: Top 10 usuarios con más accesos, distribuidos por tipo de acceso.

El panel **"ESTADÍSTICAS DE ACCESO – TEMPORAL"** muestra una información similar al panel anterior, pero detallado en el rango de tiempo seleccionado por el usuario. Se puede ver su vista previa en la Figura 168. Se compone de los siguientes bloques.

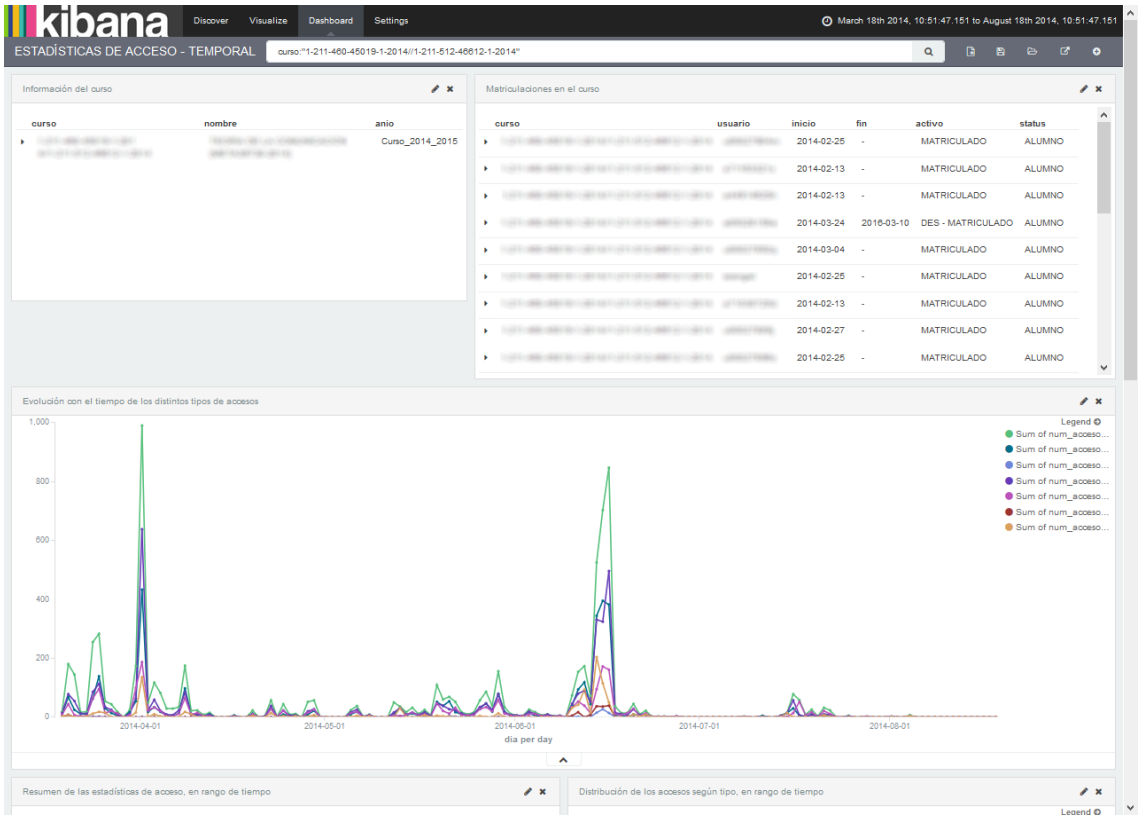


Figura 168. Vista previa del *dashboard* "ESTADÍSTICAS DE ACCESO – TEMPORAL"

- Información de curso, similar al mostrado en la Figura 110.
- Tabla de usuarios matriculados en el curso, similar a la mostrada en la Figura 159.
- Evolución con el tiempo de los distintos tipos de accesos (Figura 169).

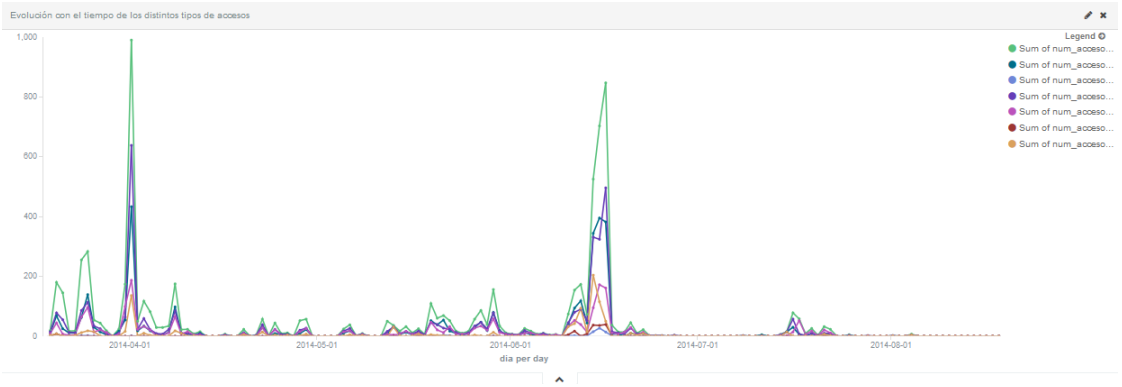


Figura 169. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO - TEMPORAL" – Visualización: Evolución con el tiempo de los distintos tipos de accesos

- Resumen de las estadísticas de acceso, en el rango de tiempo seleccionado (Figura 170).

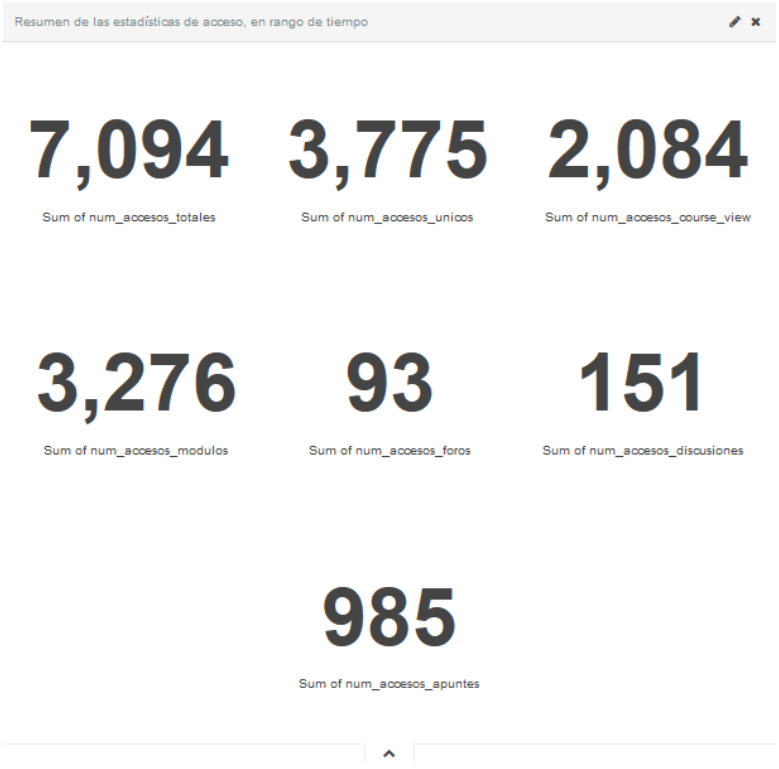


Figura 170. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO - TEMPORAL" – Visualización: Resumen de las estadísticas de acceso, en rango de tiempo

- Distribución de los accesos según tipo, en el rango de tiempo seleccionado (Figura 171).

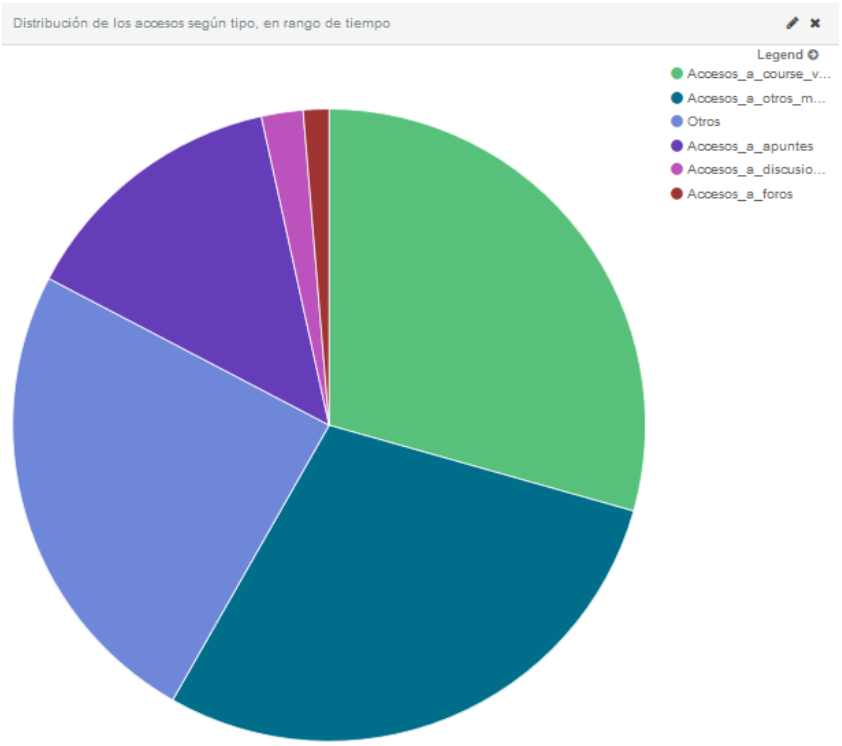


Figura 171. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO - TEMPORAL" – Visualización: Distribución de los accesos según tipo, en rango de tiempo

- Top 10 usuarios con más accesos totales, en el rango de tiempo seleccionado (Figura 172).

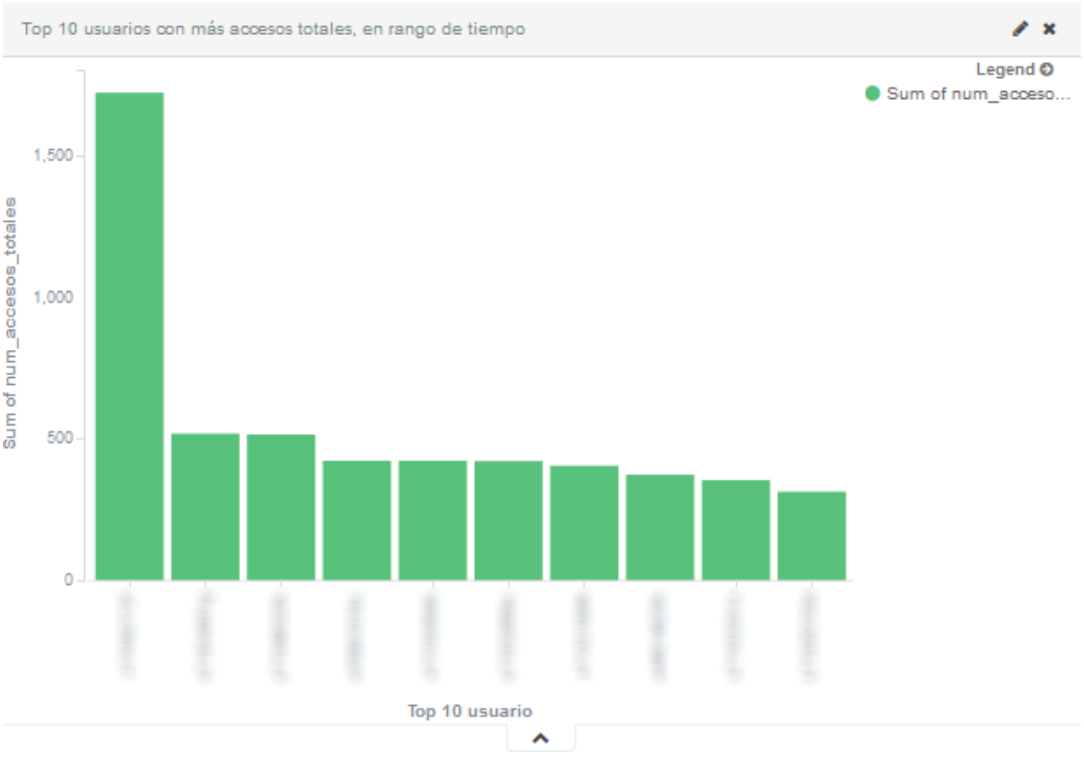


Figura 172. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO - TEMPORAL" – Visualización: Top 10 usuarios con más accesos totales, en rango de tiempo

- Distribución de los accesos según origen, en el rango de tiempo seleccionado (Figura 173).

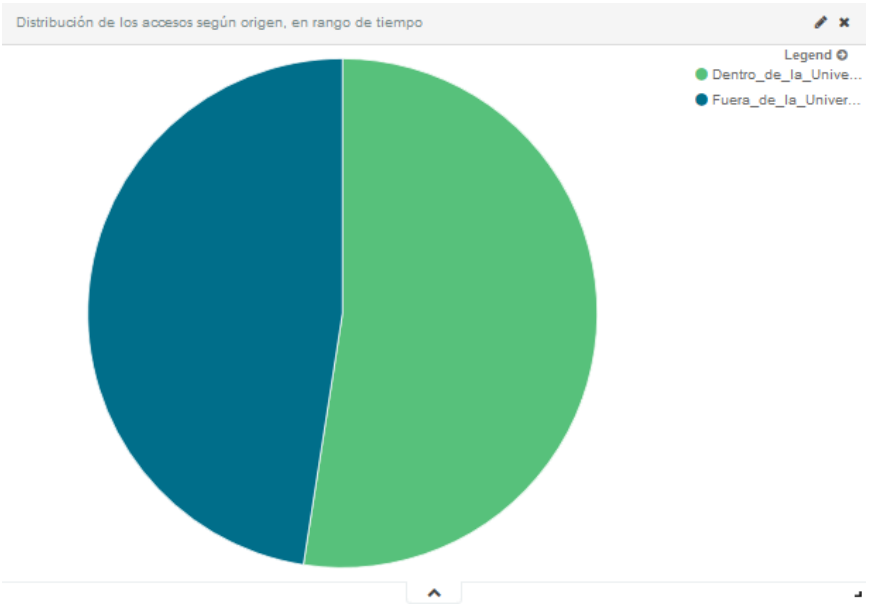


Figura 173. *Kibana*. *Dashboard* "ESTADÍSTICAS DE ACCESO - TEMPORAL" – Visualización: Distribución de los accesos según origen, en rango de tiempo.

En el panel **"ESTADÍSTICAS DE ACCESO DE UN USUARIO"** se presentan las estadísticas de acceso de un usuario concreto del curso. La Figura 174 muestra la vista previa de este panel, compuesto por los siguientes bloques.

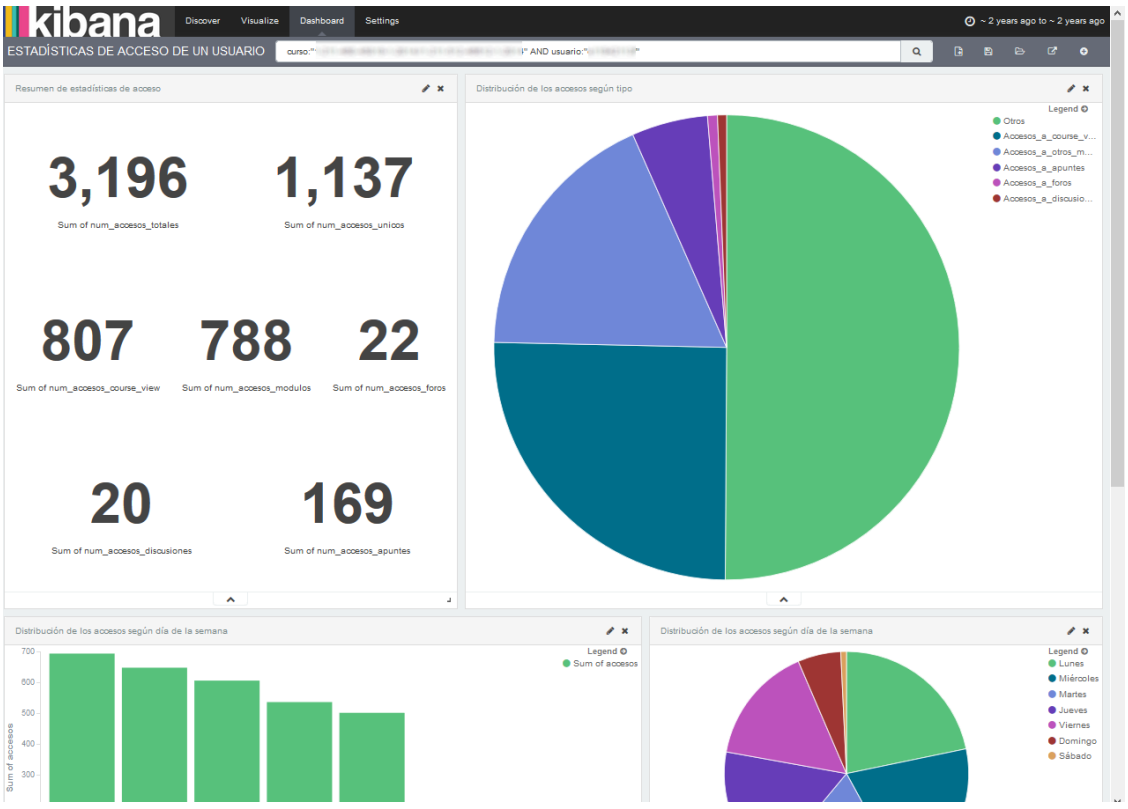


Figura 174. Vista previa del *dashboard* "ESTADÍSTICAS DE ACCESO DE UN USUARIO"

- Resumen de las estadísticas de acceso, similar al mostrado en la Figura 160, filtrando la información por usuario.
- Distribución de los accesos según tipo, similar al mostrado en la Figura 161 filtrando la información por usuario.
- Distribución de los accesos según día de la semana, similar a la Figura 162 y la Figura 163 filtrando la información por usuario.
- Distribución de los accesos según origen, similar a la Figura 164 filtrando la información por usuario.

En el panel **"ESTADÍSTICAS DE ACCESO DE UN USUARIO – TEMPORAL"** detalla las estadísticas de acceso de un usuario, en un rango de tiempo, como se puede observar en la vista previa de la Figura 175. Se compone de los siguientes bloques.

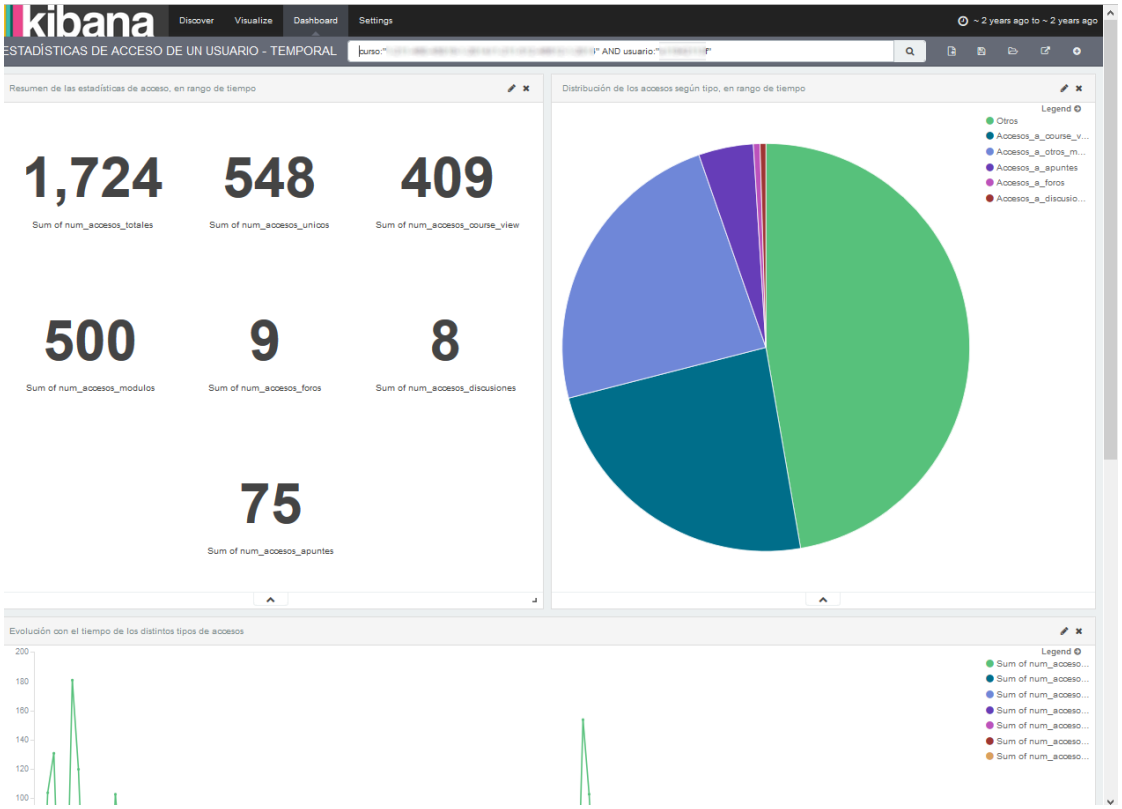


Figura 175. Vista previa del *dashboard* "ESTADÍSTICAS DE ACCESO DE UN USUARIO – TEMPORAL"

- Resumen de las estadísticas de acceso, en el rango de tiempo seleccionado. Similar al mostrado en la Figura 170 filtrando la información por usuario.
- Distribución de los accesos según tipo, en el rango de tiempo seleccionado. Similar al mostrado en la Figura 171 filtrando la información por usuario.
- Evolución con el tiempo de los distintos tipos de accesos. Similar al mostrado en la Figura 169 filtrando la información por usuario.
- Distribución de los accesos según origen, en el rango de tiempo seleccionado. Similar al mostrado en la Figura 173 filtrando la información por usuario.

En el panel **"ESTADÍSTICAS DE UN *ASSIGN*"** se presenta información sobre la actividad llevada a cabo sobre un *assign*. Se puede observar su vista previa en la Figura 176. Se compone de los siguientes bloques.

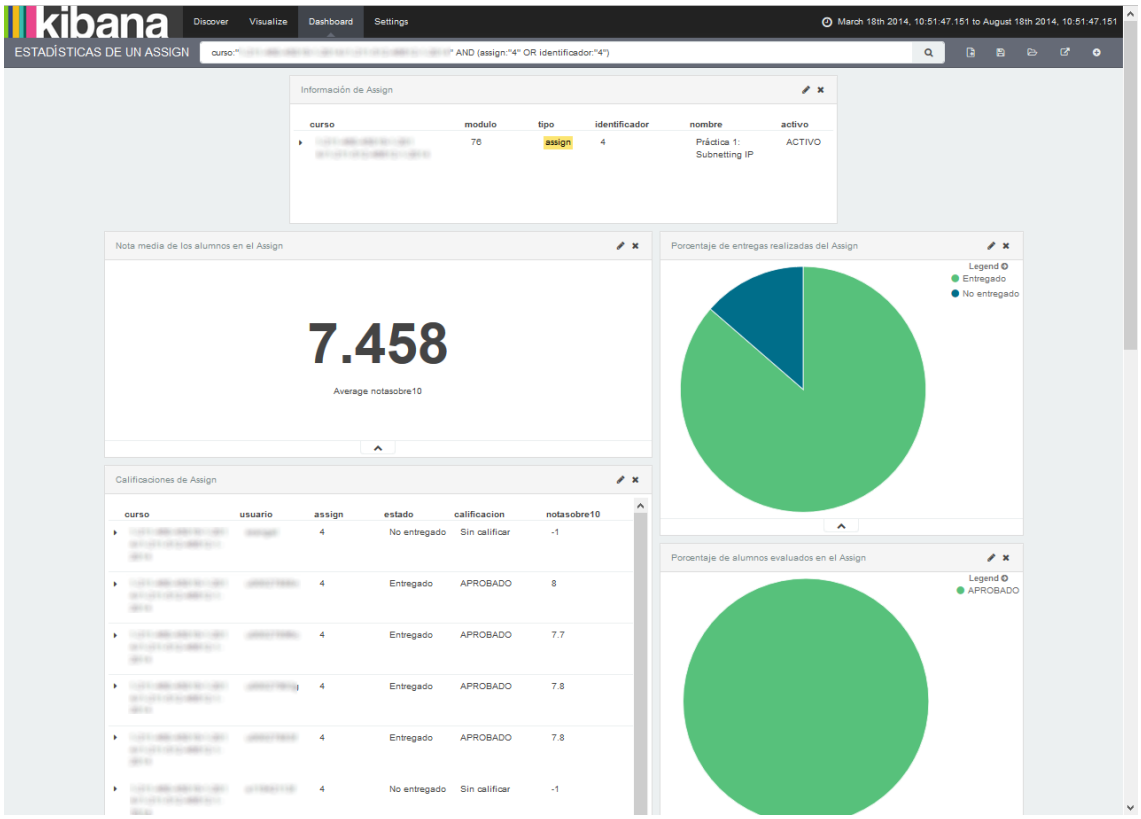


Figura 176. Vista previa del *dashboard* "ESTADÍSTICAS DE UN *ASSIGN*"

- Información del *assign* (Figura 177).

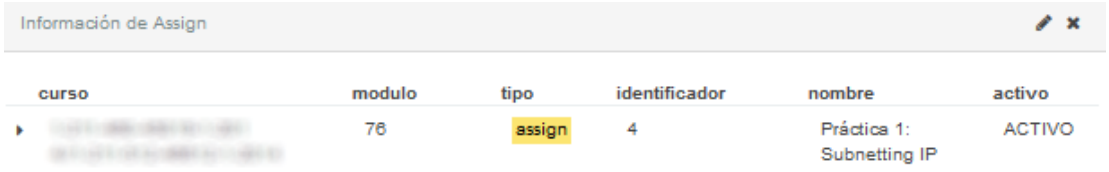


Figura 177. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *ASSIGN*" – Visualización: Información de *Assign*

- Nota media de los alumnos en el *assign* (Figura 178).

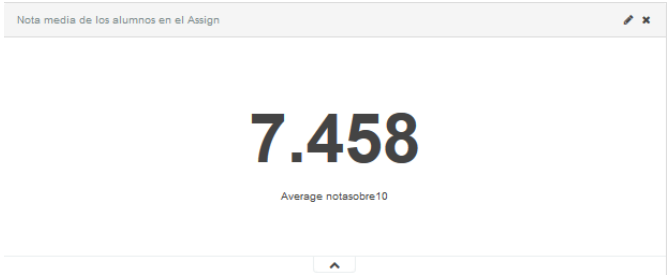


Figura 178. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *ASSIGN*" – Visualización: Nota media de los alumnos en el *Assign*

- Tabla de calificaciones en el *assign* (Figura 179).

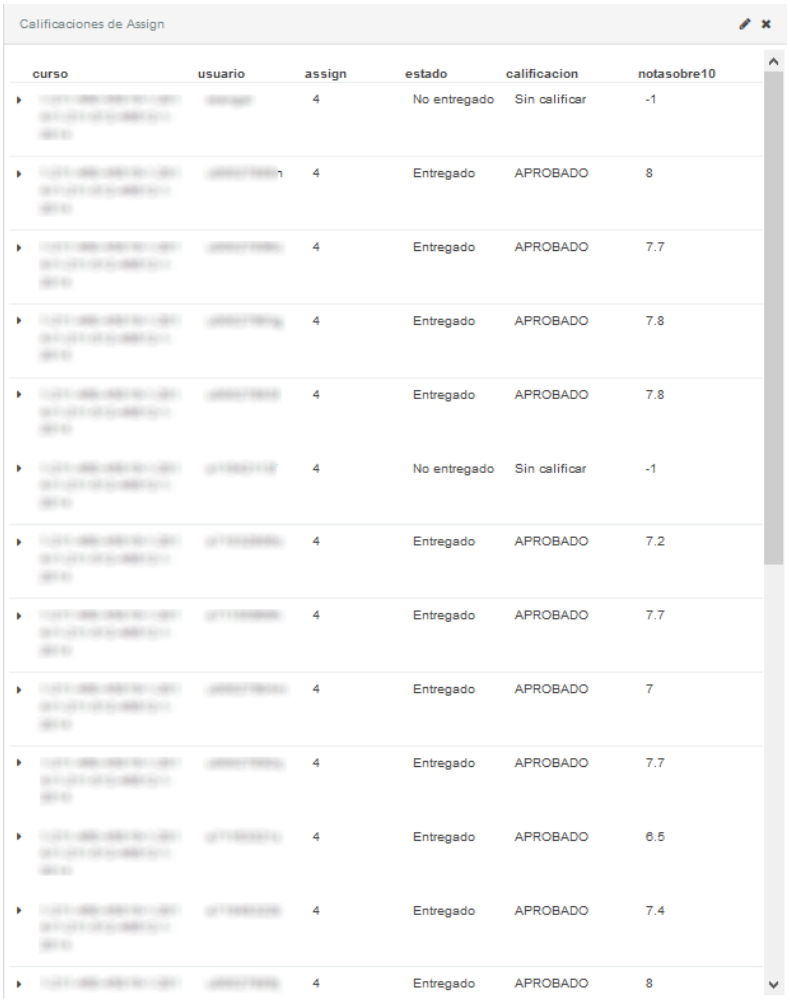


Figura 179. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *ASSIGN*" – Visualización: Calificaciones de *Assign*

- Porcentaje de entregas realizadas del *assign* (Figura 180).

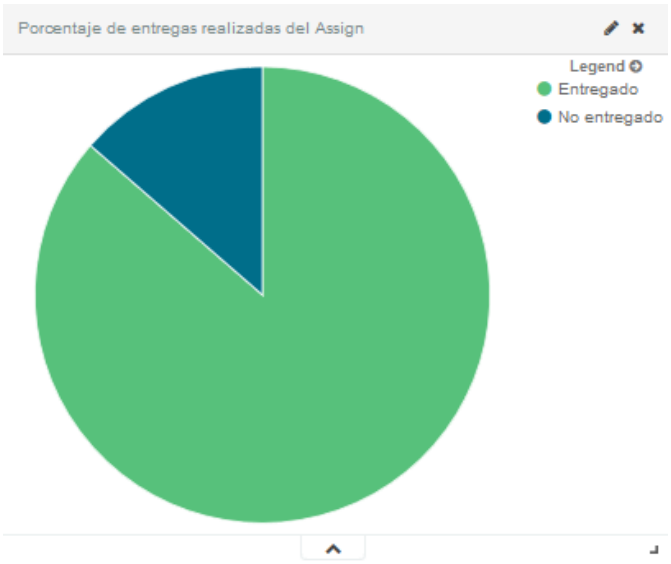


Figura 180. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *ASSIGN*" – Visualización: Porcentaje de entregas realizadas del *Assign*

- Porcentaje de alumnos evaluados en el *assign* (Figura 181).

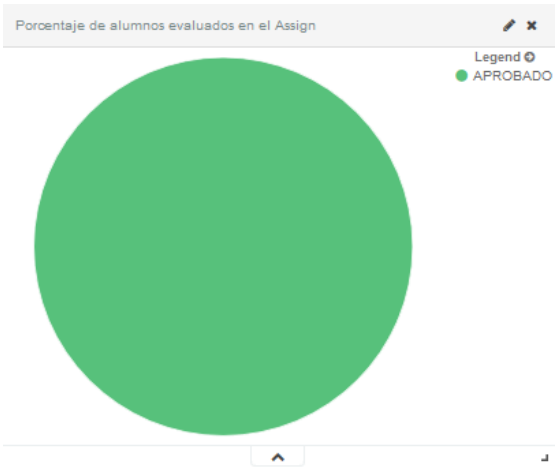


Figura 181. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *ASSIGN*" – Visualización: Porcentaje de alumnos evaluados en el *Assign*

- Porcentaje de alumnos aprobados en el *assign* (Figura 182).

Figura 182. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *ASSIGN*" – Visualización: Porcentaje de alumnos aprobados en el *Assign*

- Top 10 alumnos con mejor nota en el *assign* (Figura 183).

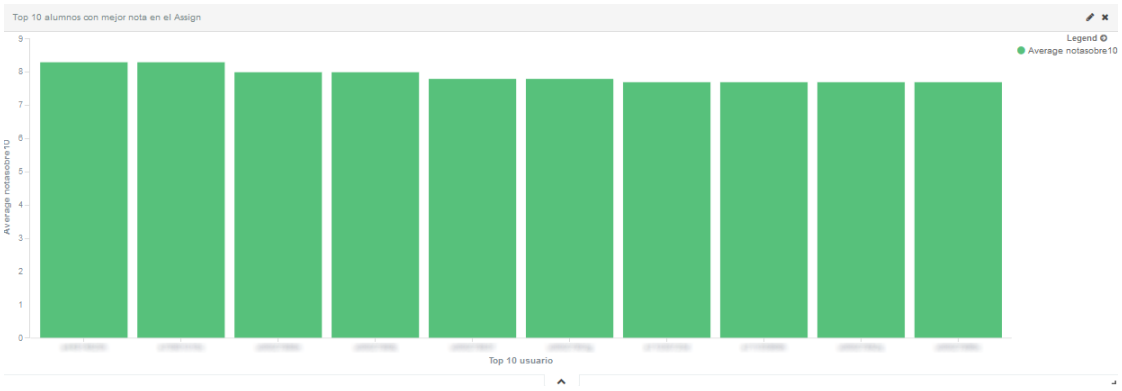


Figura 183. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *ASSIGN*" – Visualización: Top 10 alumnos con mejor nota en el *Assign*

- Correlación entre la nota en el *assign* y los accesos a los módulos, es decir, influencia del número de accesos a cada módulo sobre la nota que cada alumno ha obtenido en el *assign* (Figura 184).

Figura 184. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *ASSIGN*" – Visualización: Correlación Nota en el *Assign* – Acceso a Módulos

- Representación gráfica de la relación anterior (Figura 185).

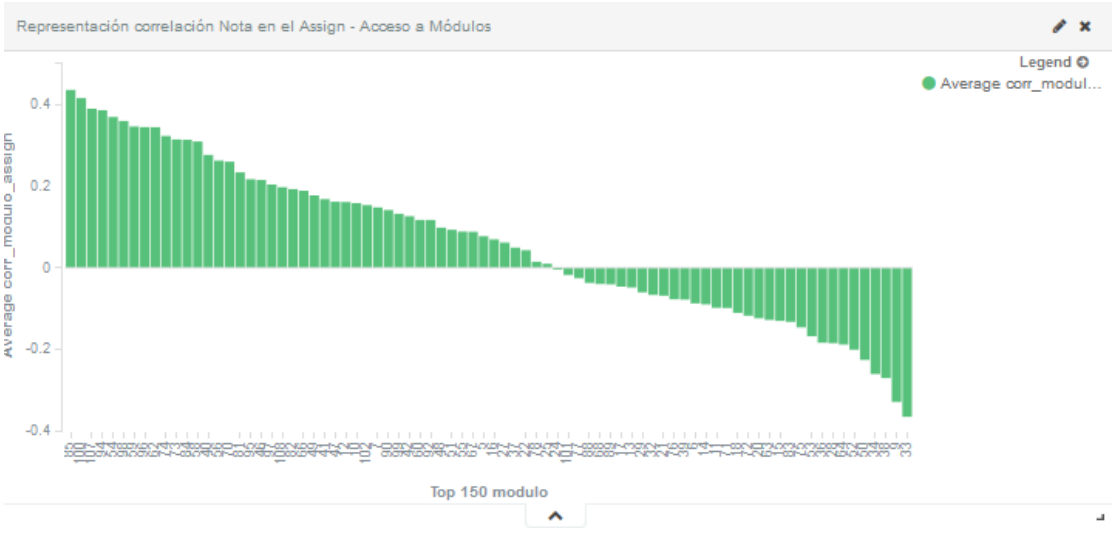


Figura 185. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *ASSIGN*" – Visualización: Representación correlación Nota en el *Assign* – Acceso a Módulos

En el panel **"ESTADÍSTICAS DE UN *QUIZ*"** se presenta información sobre la actividad llevada a cabo sobre un *quiz*. Se puede observar su vista previa en la Figura 186. Se compone de los bloques que se enumeran a continuación.

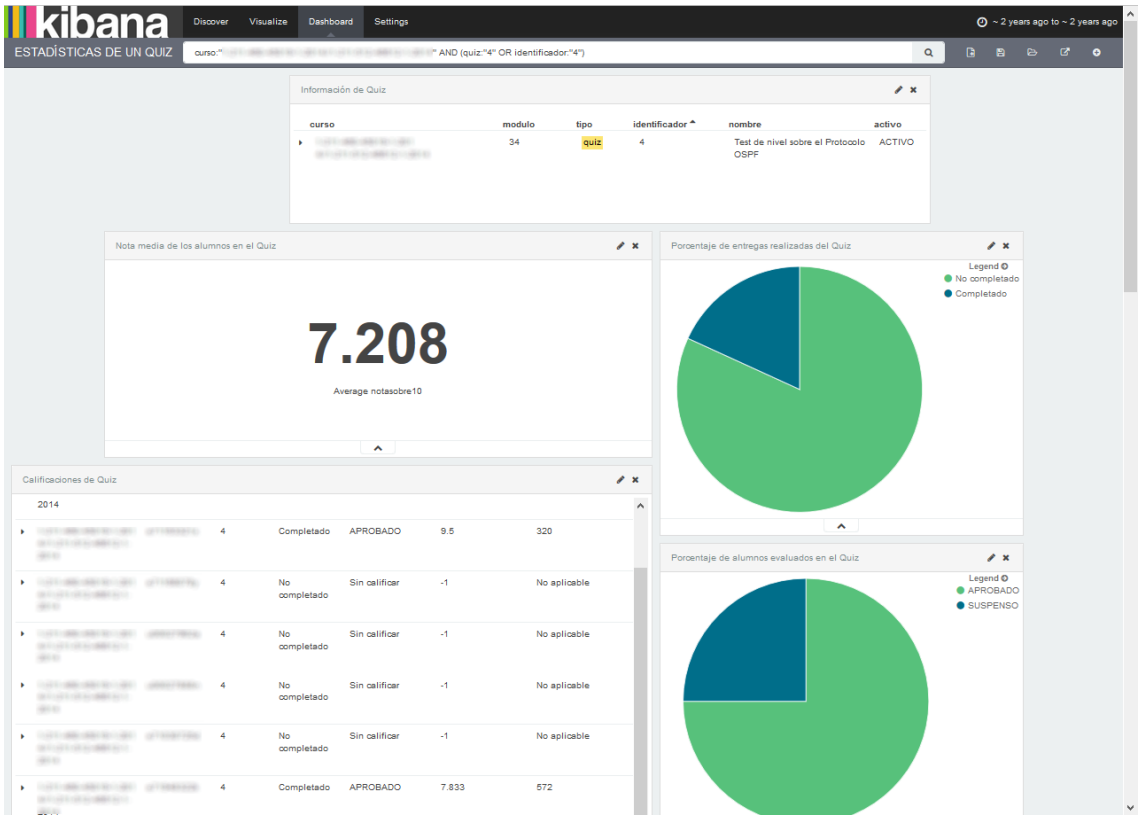


Figura 186. Vista previa del *dashboard* "ESTADÍSTICAS DE UN *QUIZ*"

- Información del *quiz* (Figura 187).

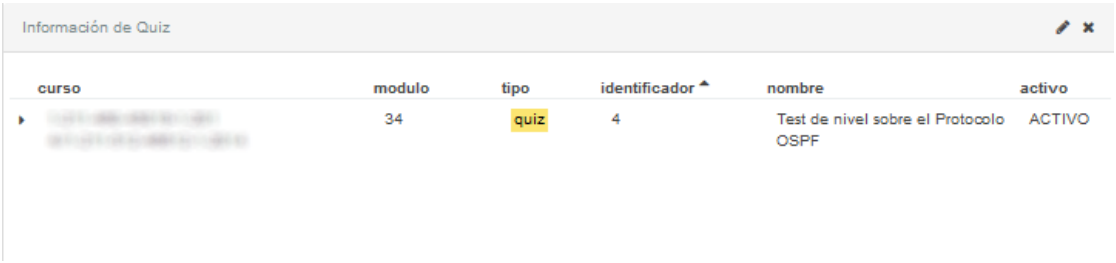


Figura 187. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *QUIZ*" – Visualización: Información de *Quiz*

- Nota media de los alumnos en el *quiz* (Figura 188).

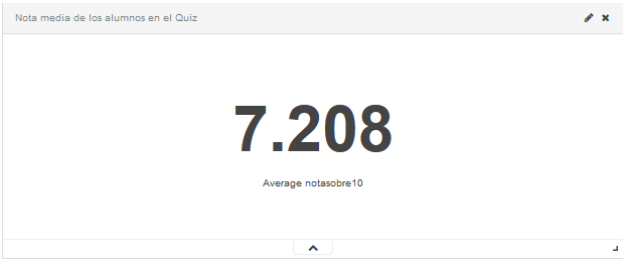


Figura 188. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *QUIZ*" – Visualización: Nota media de los alumnos en el *Quiz*

Figura 189. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *QUIZ*" – Visualización: Calificaciones de *Quiz*

- Tabla de calificaciones en el *quiz* (Figura 189).
- Porcentaje de entregas realizadas del *quiz* (Figura 190).

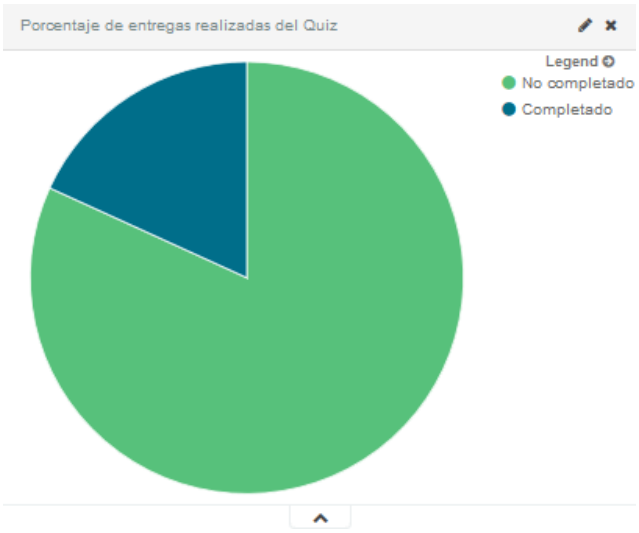


Figura 190. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *QUIZ*" – Visualización: Porcentaje de entregas realizadas del *Quiz*

- Porcentaje de alumnos evaluados en el *quiz* (Figura 191).

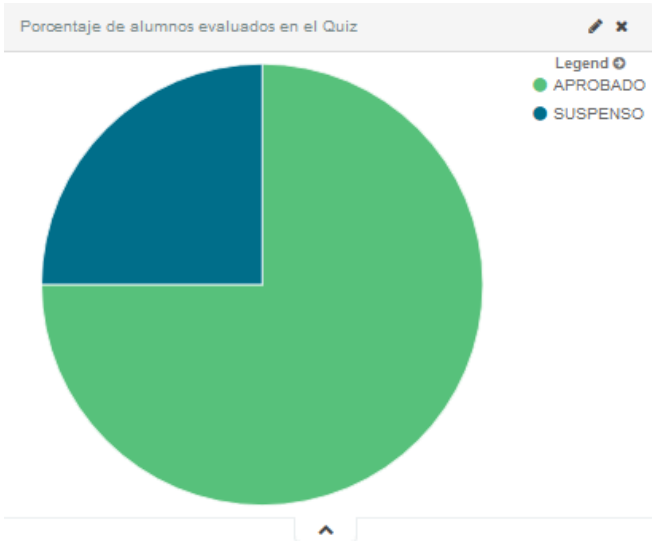


Figura 191. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *QUIZ*" – Visualización: Porcentaje de alumnos evaluados en el *Quiz*

- Porcentaje de alumnos aprobados en el *quiz* (Figura 192).

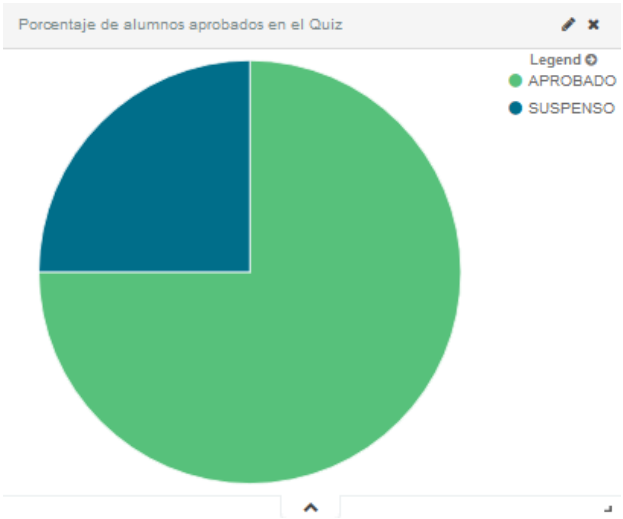


Figura 192. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *QUIZ*" – Visualización: Porcentaje de alumnos aprobados en el *Quiz*

- Top 10 alumnos con mejor nota en el *quiz* (Figura 193).

Figura 193. Kibana. Dashboard "ESTADÍSTICAS DE UN QUIZ" – Visualización: Top 10 alumnos con mejor nota en el *Quiz*

- Top 5 alumnos que menos han tardado en realizar el *quiz* (Figura 194).

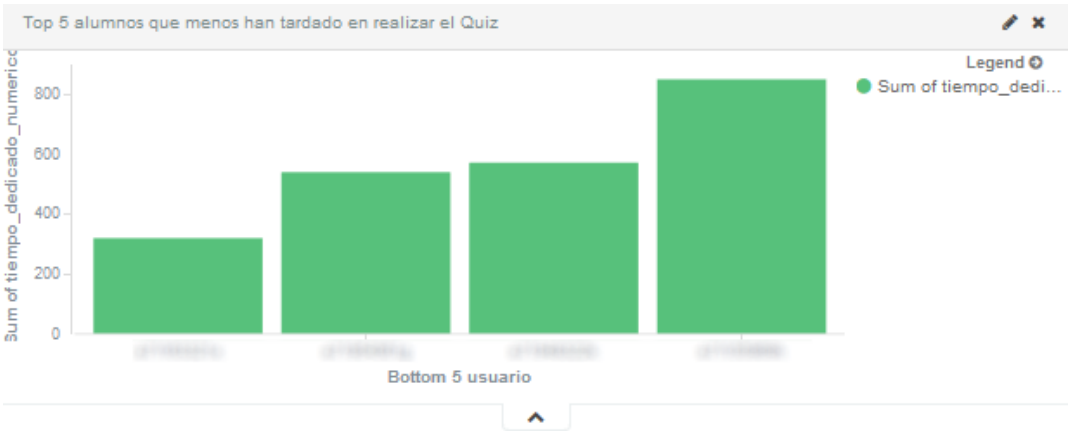


Figura 194. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *QUIZ*" – Visualización: Top 5 alumnos que menos han tardado en realizar el *Quiz*

- Top 5 alumnos que más han tardado en realizar el *quiz* (Figura 195).

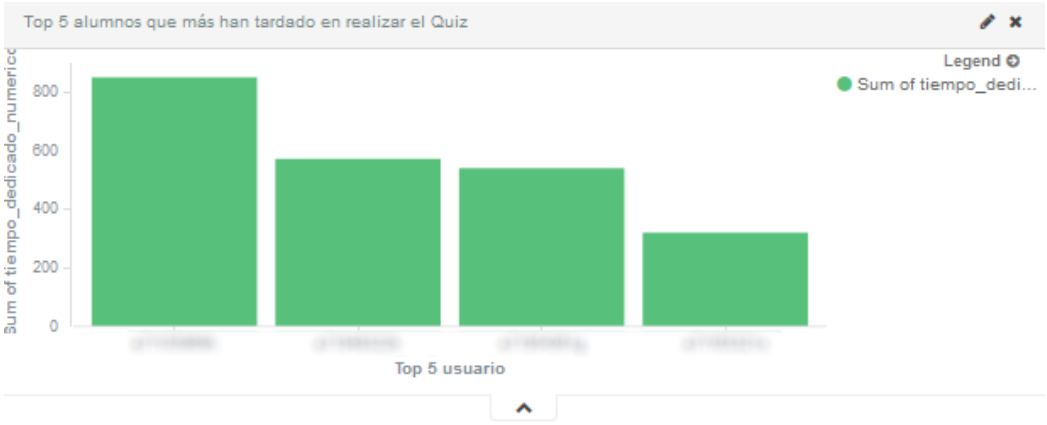


Figura 195. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *QUIZ*" – Visualización: Top 5 alumnos que más han tardado en realizar el *Quiz*

Figura 196. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *QUIZ*" – Visualización: Correlación Nota en el *Quiz* – Acceso a Módulos

- Correlación entre la nota en el *quiz* y los accesos a los módulos, es decir, influencia entre el número de accesos a cada módulo sobre la nota que cada alumno ha obtenido en el *quiz* (Figura 196).

- Representación gráfica de la relación anterior (Figura 197).

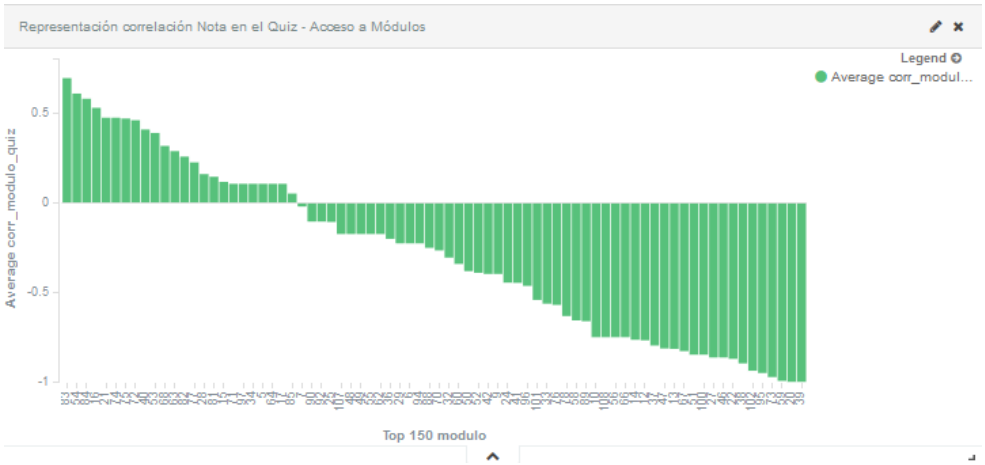


Figura 197. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *QUIZ*" – Visualización: Representación correlación Nota en el *Quiz* – Acceso a Módulos

- Influencia del tiempo dedicado en la realización del *quiz* sobre la nota obtenida (Figura 198).

Figura 198. *Kibana*. *Dashboard* "ESTADÍSTICAS DE UN *QUIZ*" – Visualización: Influencia del tiempo dedicado en la realización del *Quiz* sobre su nota

En el panel **"ESTADÍSTICAS *QUIZ*"** se representa la actividad realizada sobre los *quiz* de un curso, como se muestra en la Figura 199. Se compone de las siguientes visualizaciones.

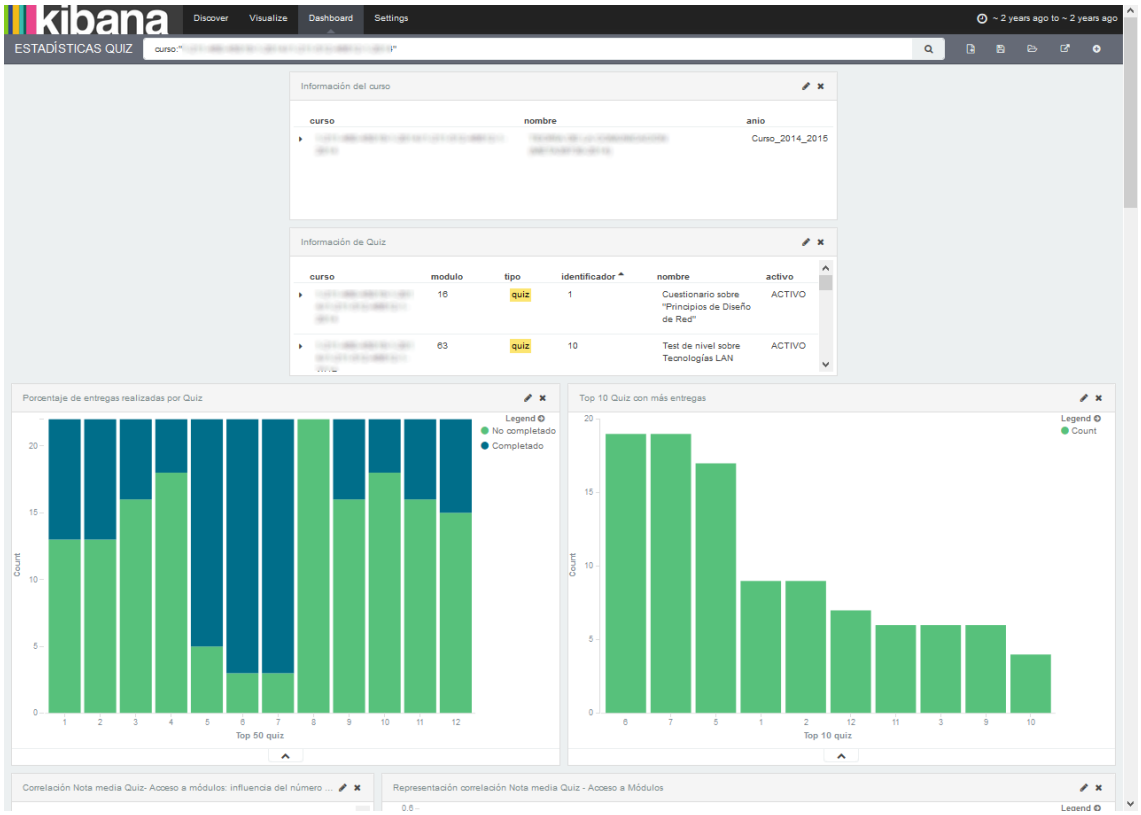


Figura 199. Vista previa del *dashboard* "ESTADÍSTICAS *QUIZ*"

- Información del curso, similar al mostrado en la Figura 110.
- Información de *quiz* publicados en el curso (Figura 200).

Figura 200. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Información de *Quiz*

- Porcentaje de entregas realizadas por *quiz* (Figura 201).

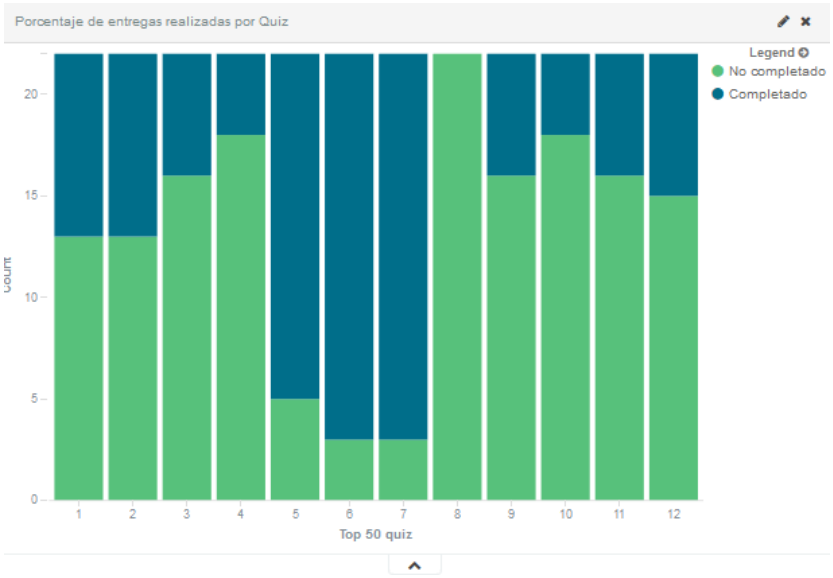


Figura 201. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Porcentaje de entregas realizadas por *Quiz*

- Top 10 *quiz* con más entregas (Figura 202).

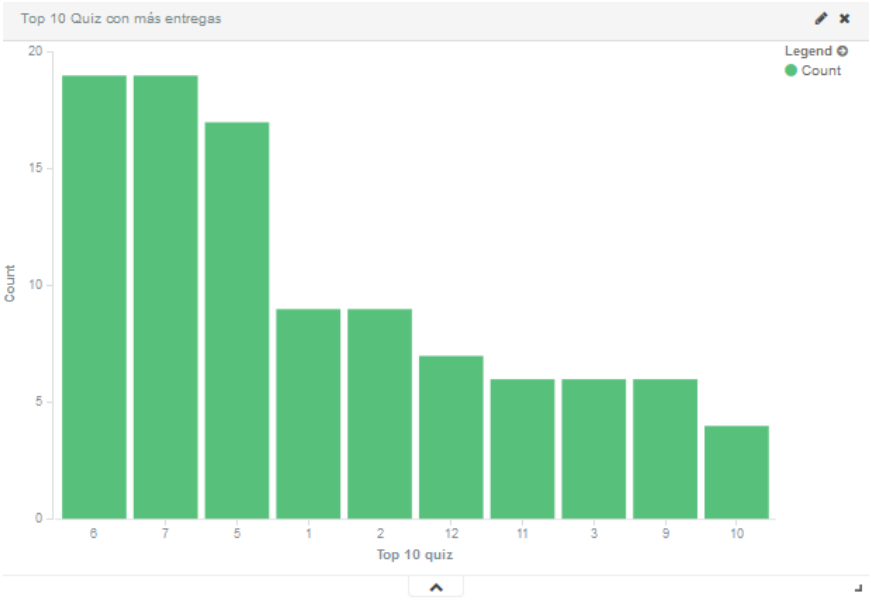


Figura 202. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Top 10 *Quiz* con más entregas

- Correlación entre la nota media obtenida en *quiz* y los accesos a los módulos, es decir, influencia del número de accesos a cada módulo sobre la nota media obtenida por cada alumno en los *quiz* del curso (Figura 203).

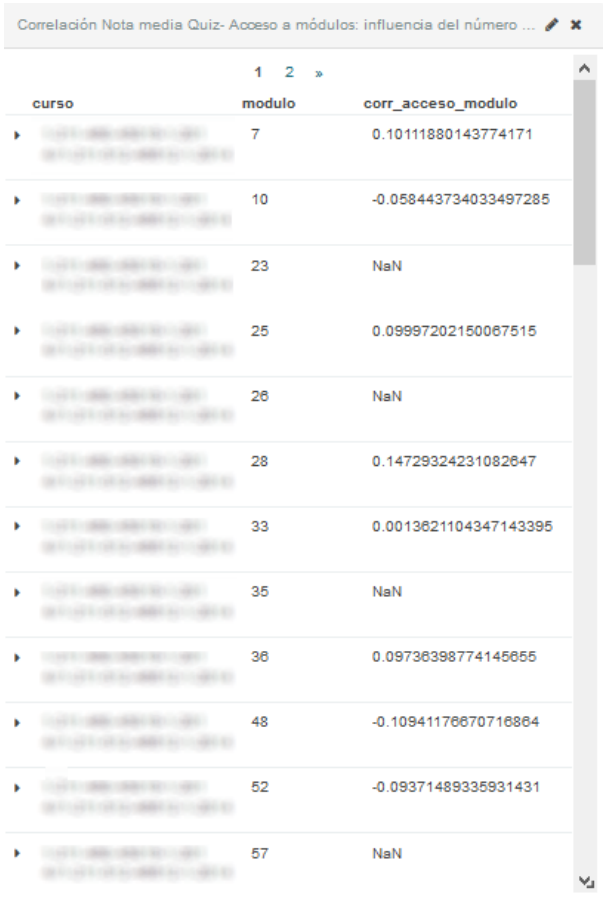


Figura 203. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Correlación Nota media *Quiz* – Acceso a Módulos

- Representación gráfica de la relación anterior (Figura 204).

Figura 204. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Representación correlación Nota media *Quiz* – Acceso a Módulos

- Porcentaje de relaciones *quiz* – alumno calificados en el curso (Figura 205).

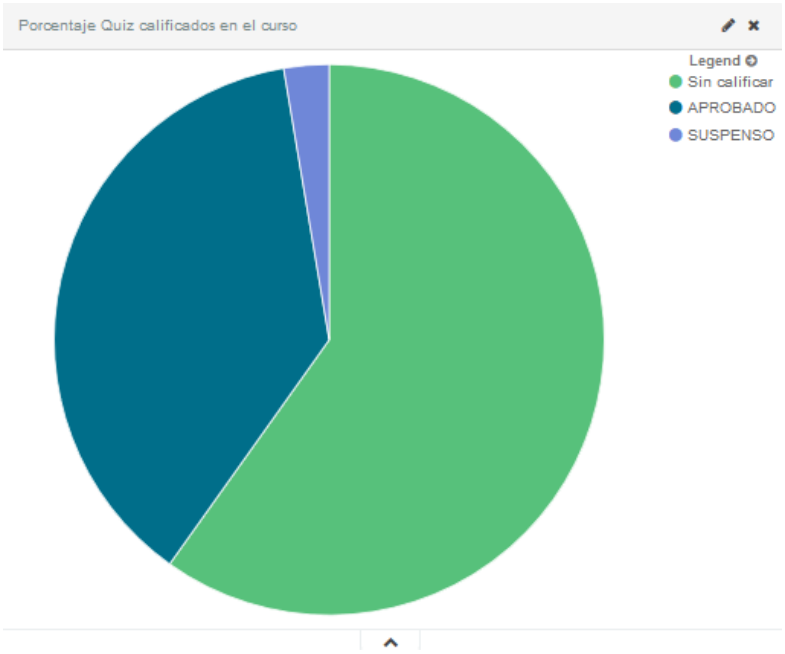


Figura 205. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Porcentaje *Quiz* calificados en el curso

- Calificaciones realizadas por *quiz* (Figura 206).

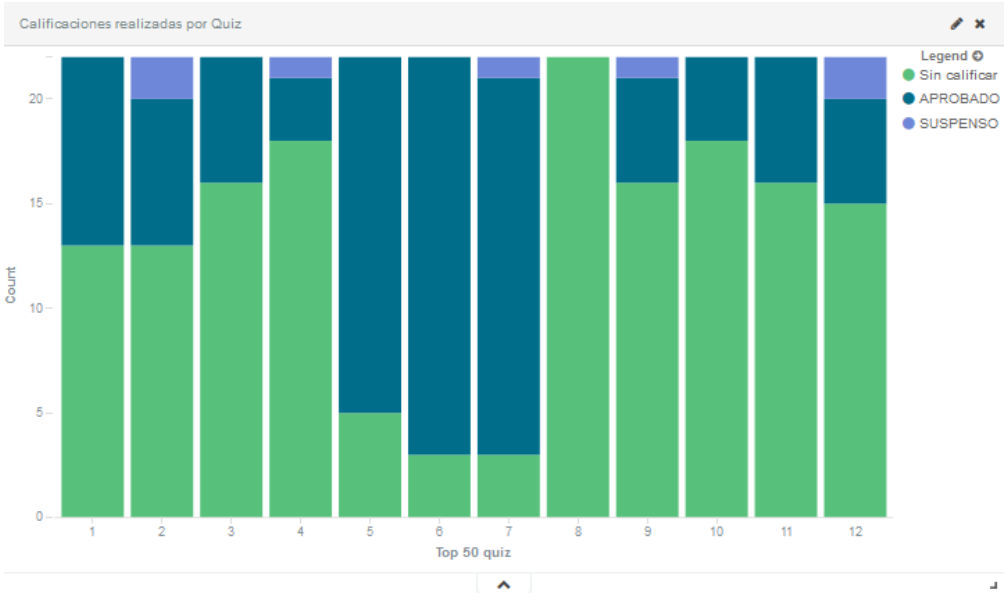


Figura 206. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Calificaciones realizadas por *Quiz*

- Top 10 *quiz* con mayor nota media (Figura 207).

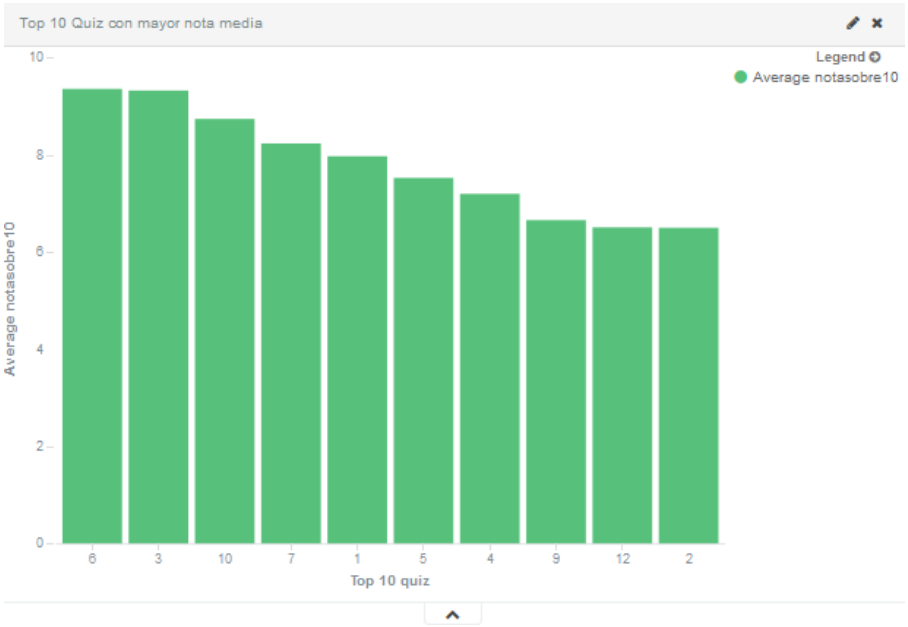


Figura 207. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Top 10 *Quiz* con mayor nota media

- Top 10 alumnos con mejor nota media en *quiz* (Figura 208).

Figura 208. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Top 10 alumnos con mejor nota media en *Quiz*

- Top 10 *quiz* con mayor tiempo promedio de realización (Figura 209).

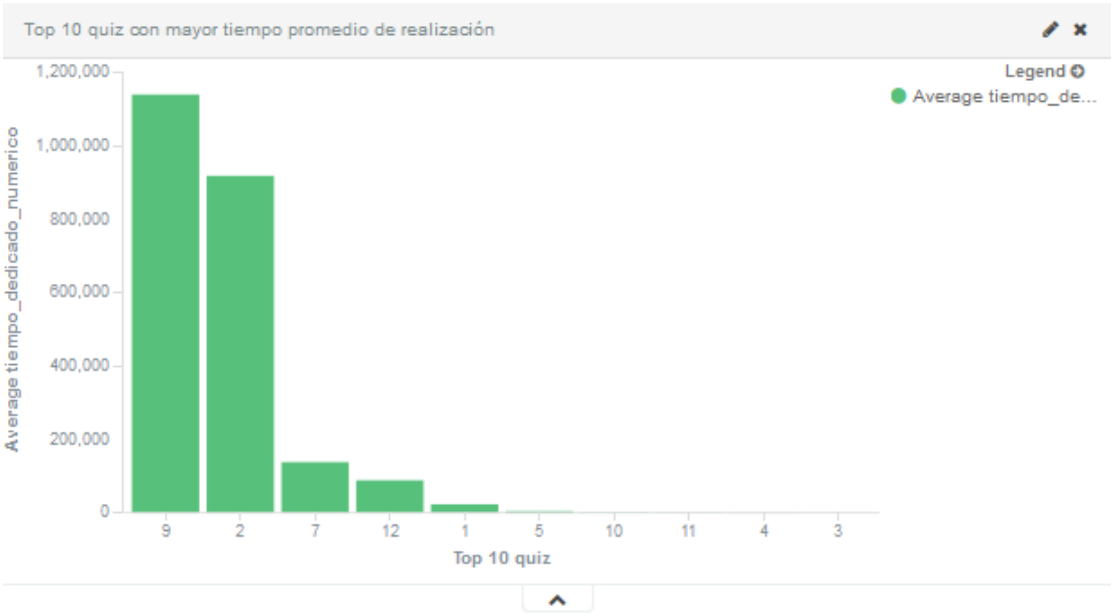


Figura 209. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Top 10 *Quiz* con mayor tiempo promedio de realización

- Top 10 *quiz* con menos tiempo promedio de realización (Figura 210).

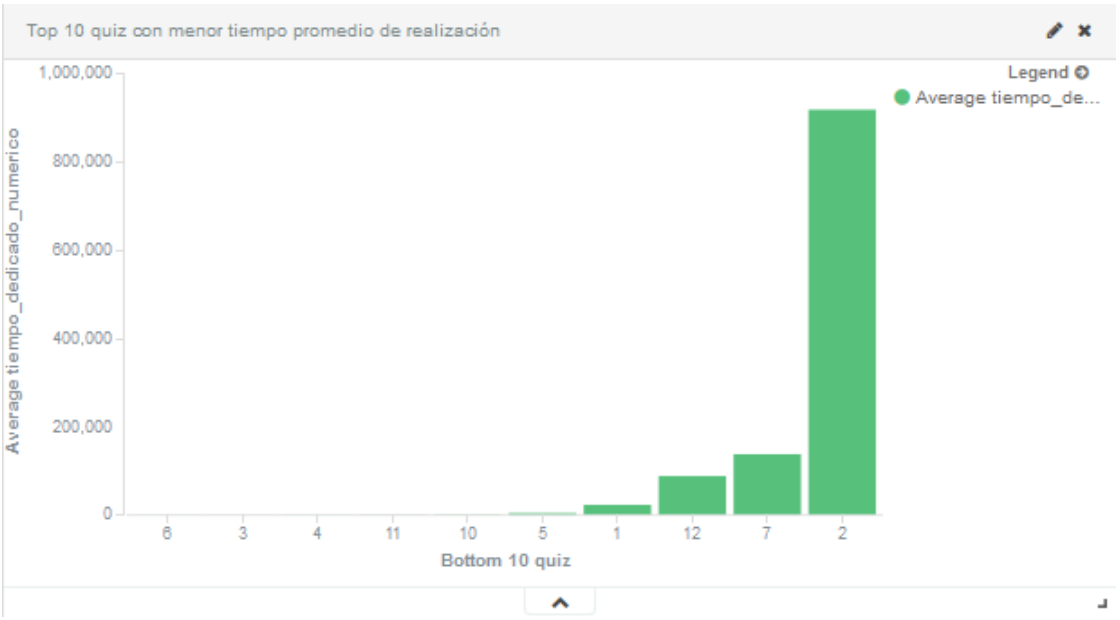


Figura 210. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Top 10 *Quiz* con menos tiempo promedio de realización

- Distribución de las calificaciones en *quiz* realizadas por alumno (Figura 211).

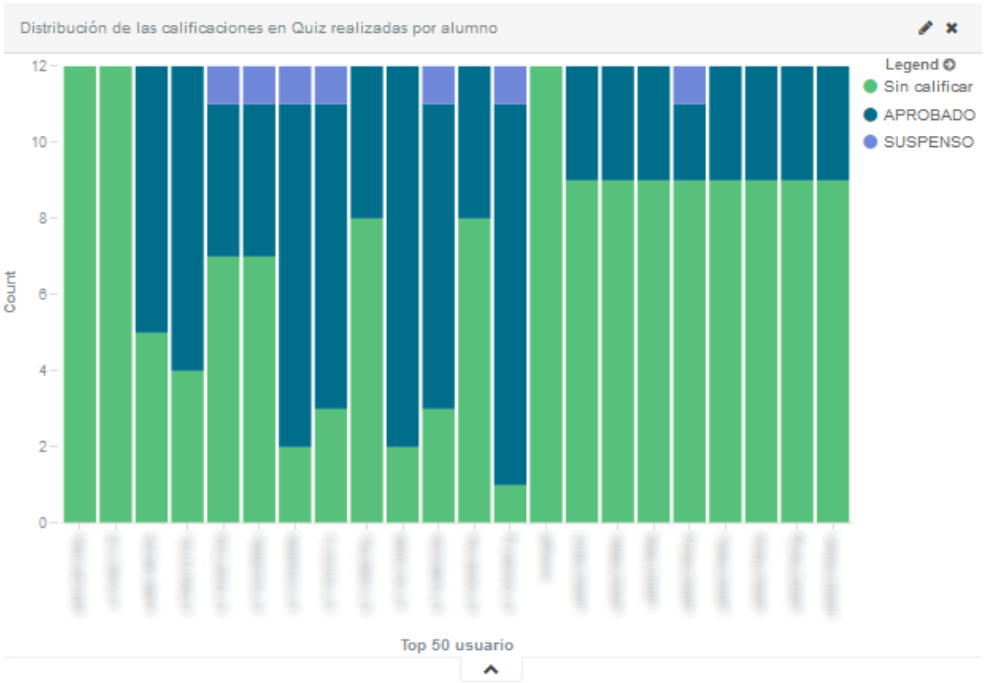


Figura 211. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Distribución de las calificaciones en *Quiz* realizadas por alumno

- Porcentaje de entregas de *quiz* por alumno (Figura 212).

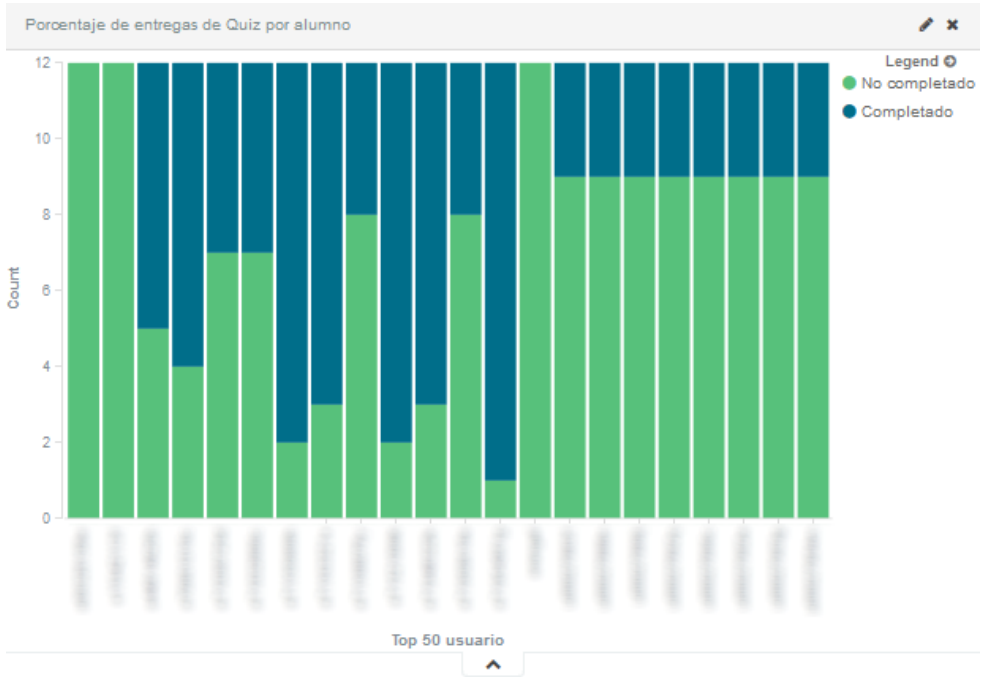


Figura 212. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Porcentaje de entregas de *Quiz* por alumno

- Influencia del tiempo dedicado en *quiz* sobre la nota final y sobre la nota media en *quiz* (Figura 213).

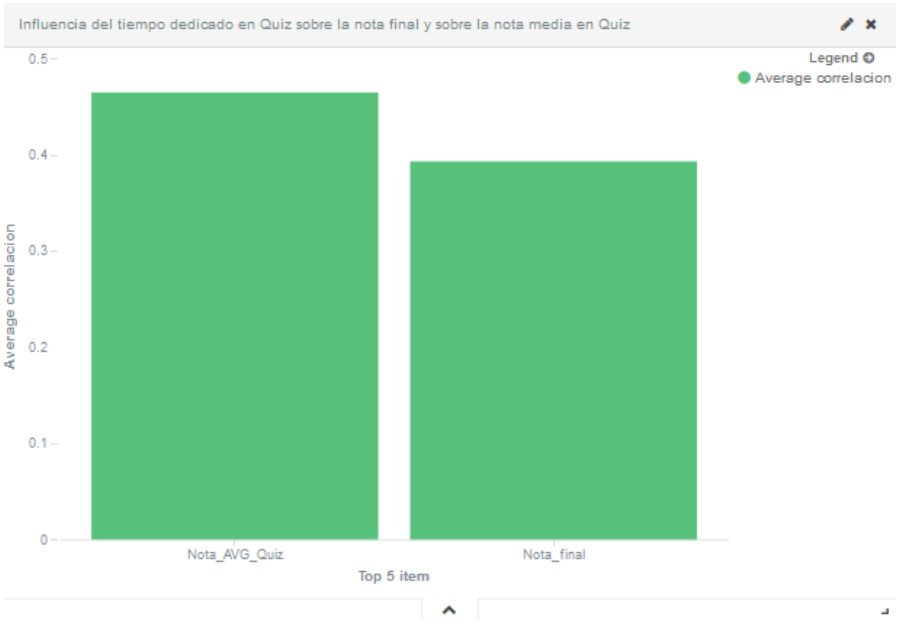


Figura 213. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ*" – Visualización: Influencia de tiempo dedicado en *Quiz* sobre la nota final y sobre la nota media en *Quiz*

En el panel **"ESTADÍSTICAS *QUIZ* – USUARIO"** se representa las estadísticas de actividad de un alumno, sobre los *quiz* de un curso. Se puede observar su vista previa en la Figura 214. Se compone de los siguientes bloques.

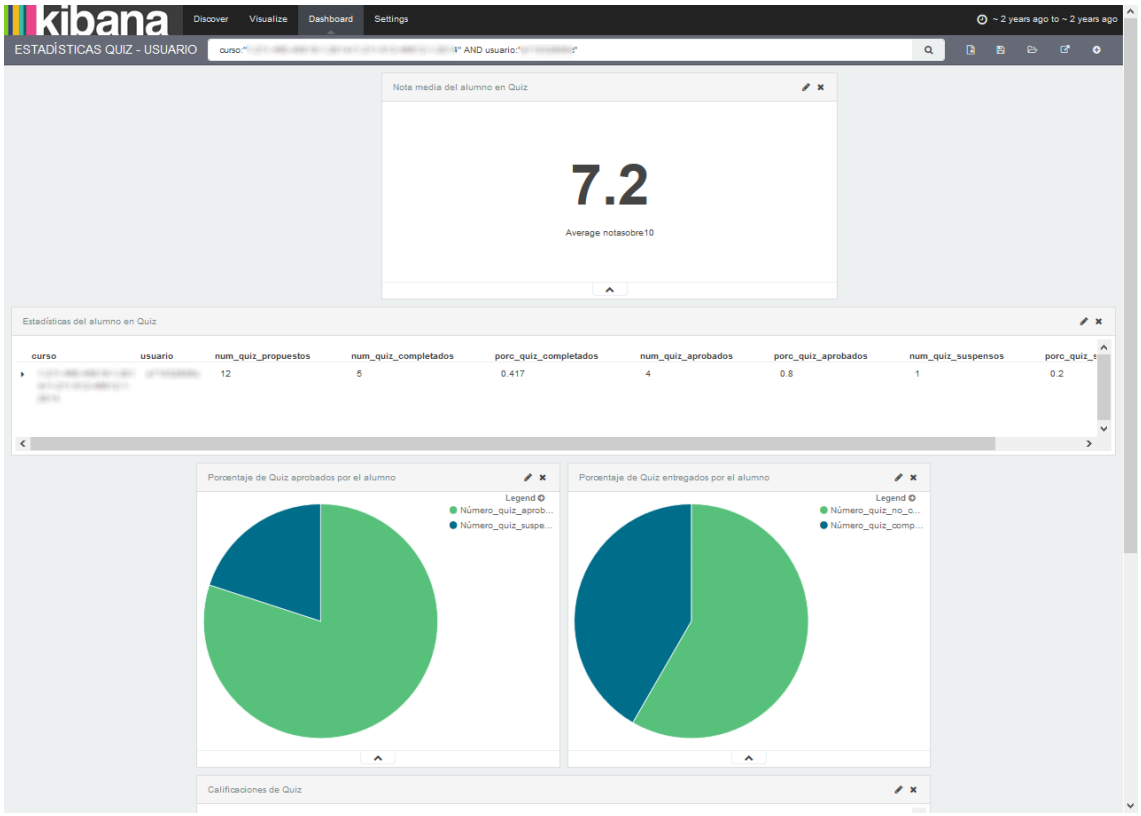


Figura 214. Vista previa del *dashboard* "ESTADÍSTICAS *QUIZ* – USUARIO"

- Nota media del alumno en *quiz* (Figura 215).

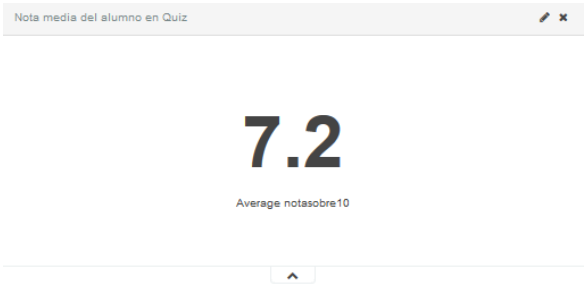


Figura 215. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ* - USUARIO" – Visualización: Nota media del alumno en *Quiz*

- Estadísticas del alumno en *quiz* (Figura 216).

Figura 216. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ* - USUARIO" – Visualización: Estadísticas del alumno en *Quiz*

- Porcentaje de *quiz* aprobados por el alumno (Figura 217).

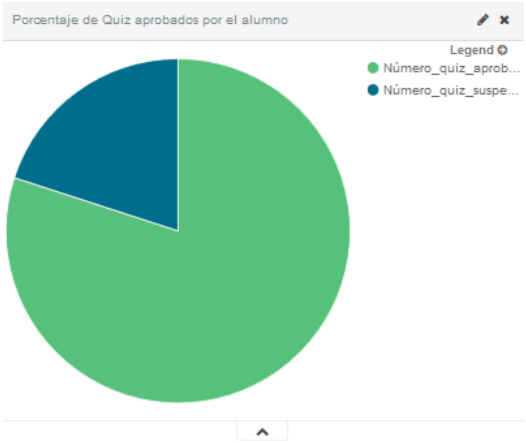


Figura 217. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ* - USUARIO" – Visualización: Porcentaje de *Quiz* aprobados por el alumno

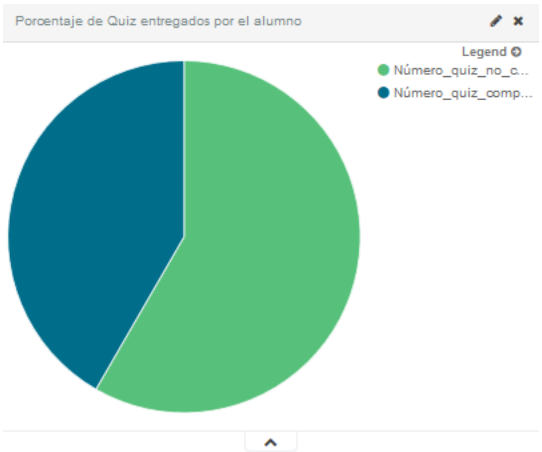


Figura 218. *Kibana*. *Dashboard* "ESTADÍSTICAS QUIZ - USUARIO" – Visualización: Porcentaje de *Quiz* entregados por el alumno

- Porcentaje de *quiz* entregados por el alumno (Figura 218).
- Tabla de calificaciones del alumno en *quiz* (Figura 219).

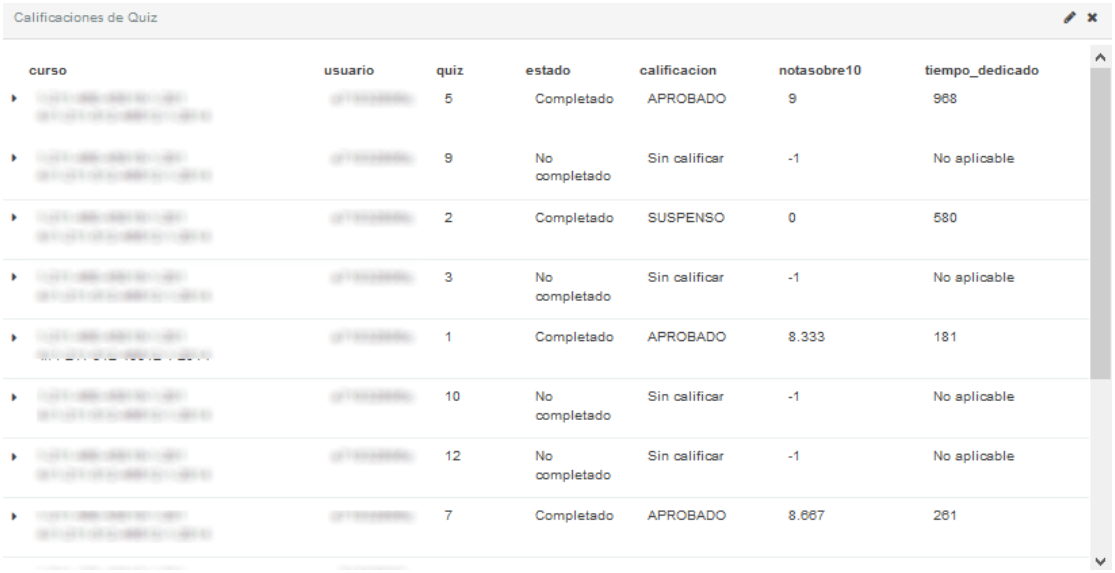


Figura 219. *Kibana*. *Dashboard* "ESTADÍSTICAS *QUIZ* - USUARIO" – Visualización: Calificaciones de *Quiz*

El panel **"ESTADÍSTICAS SOBRE EL USO DE APUNTES"** muestra información sobre cómo se accede a los apuntes del curso, como se puede observar en la Figura 220. Podemos distinguir las siguientes vistas.

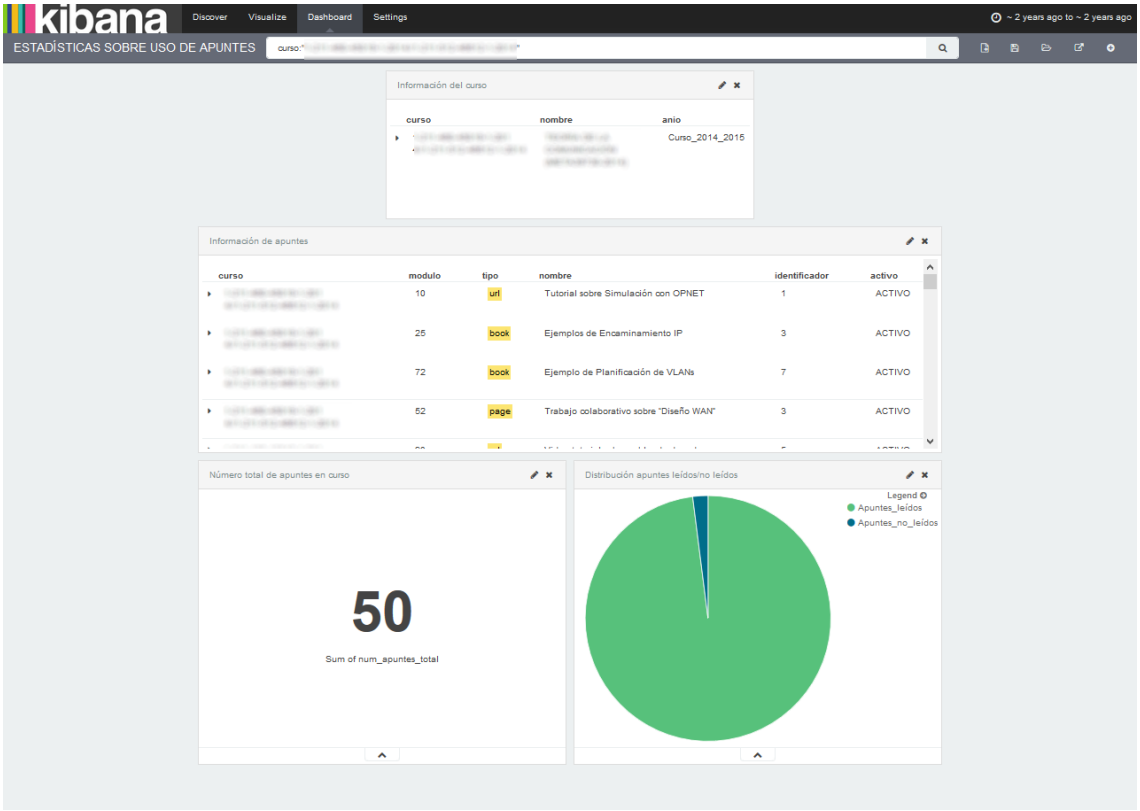


Figura 220. Vista previa del *dashboard* "ESTADÍSTICAS SOBRE USO DE APUNTES"

- Información del curso, similar al mostrado en la Figura 110.

- Información de apuntes publicados en el curso (Figura 221).

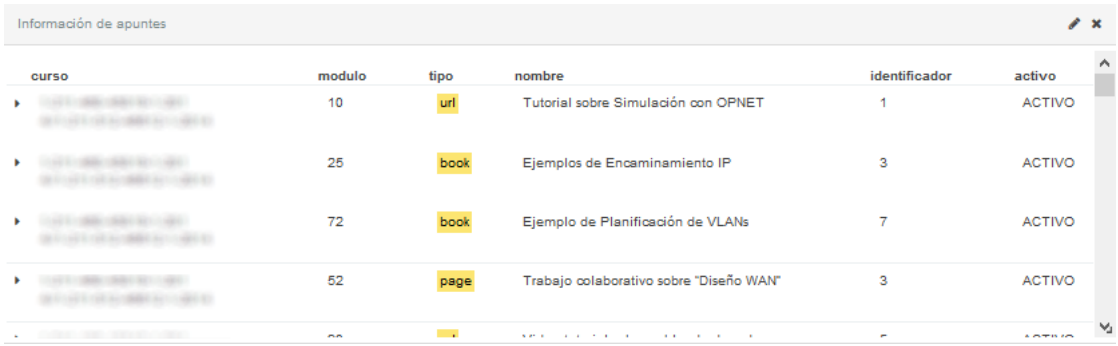


Figura 221. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE APUNTES" – Visualización: Información de apuntes

- Número total de apuntes en el curso (Figura 222).

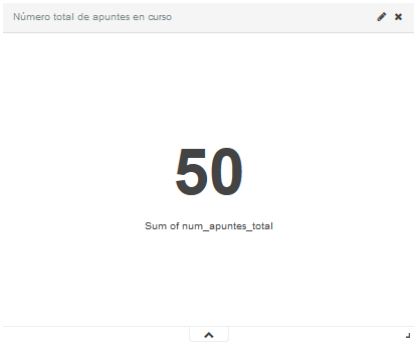


Figura 222. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE APUNTES" – Visualización: Número total de apuntes en curso

- Distribución de apuntes leídos / no leídos (Figura 223).

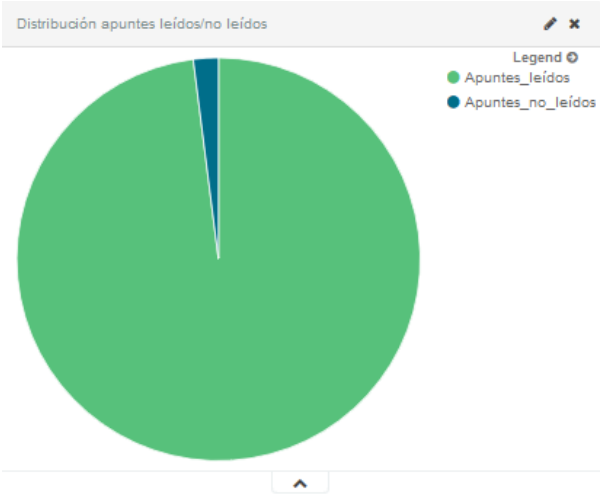


Figura 223. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE APUNTES" – Visualización: Distribución apuntes leídos / no leídos

El panel **"ESTADÍSTICAS SOBRE USO DE FOROS"** muestra información sobre cómo los usuarios usan los foros creados en el curso. Como se puede observar en la Figura 224, proporciona las siguientes vistas.

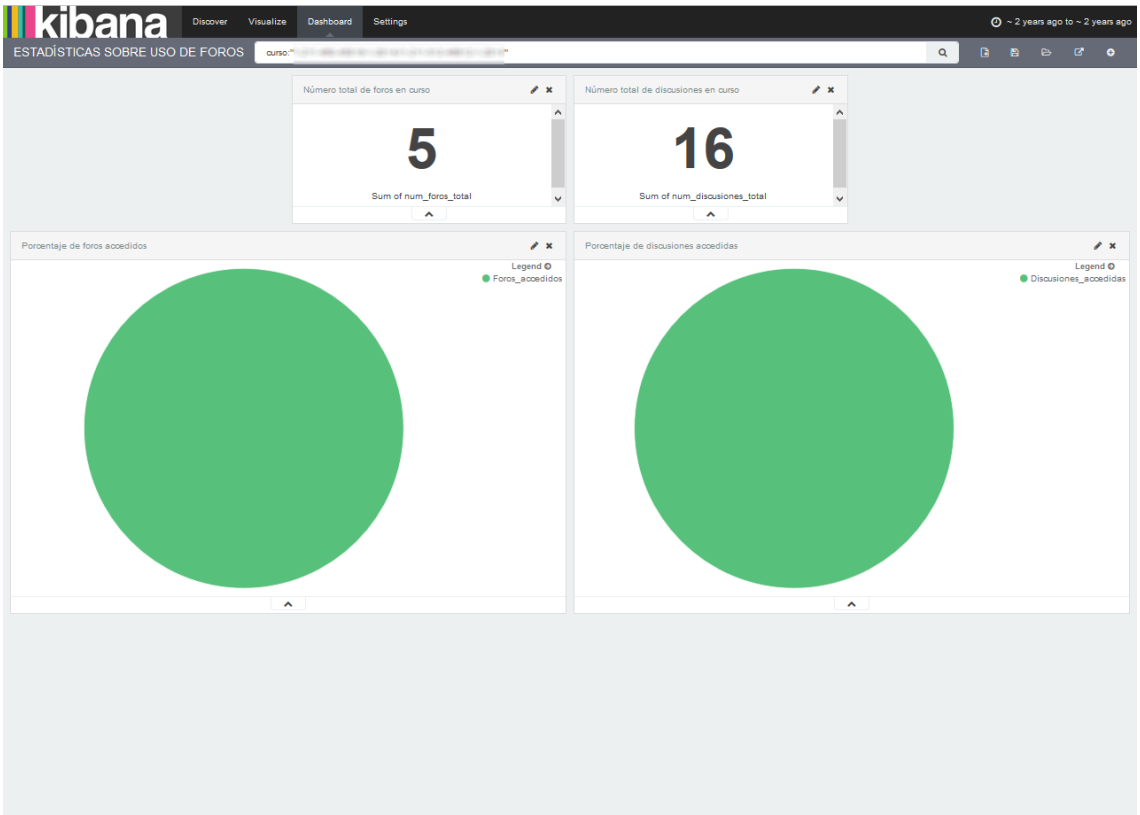


Figura 224. Vista previa del *dashboard* "ESTADÍSTICAS SOBRE USO DE FOROS"

- Número total de foros en el curso (Figura 225).

Figura 225. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE FOROS" – Visualización: Número total de foros en curso

- Número total de discusiones en el curso (Figura 226).

Figura 226. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE FOROS" – Visualización: Número total de discusiones en curso

- Porcentaje de foros accedidos (Figura 227).

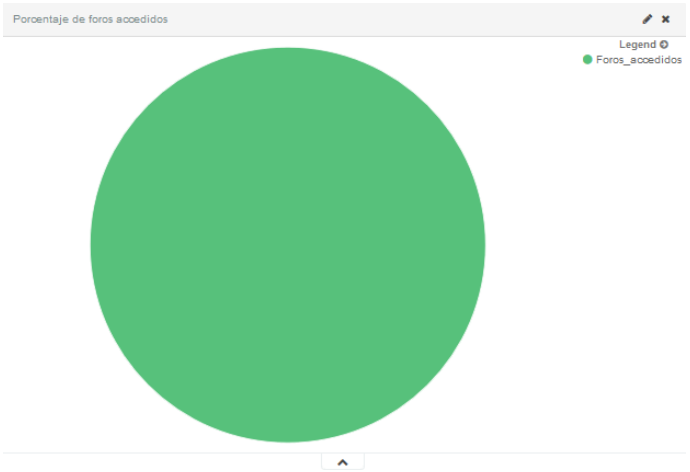


Figura 227. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE FOROS" – Visualización: Porcentaje de foros accedidos

- Porcentaje de discusiones accedidas (Figura 228).

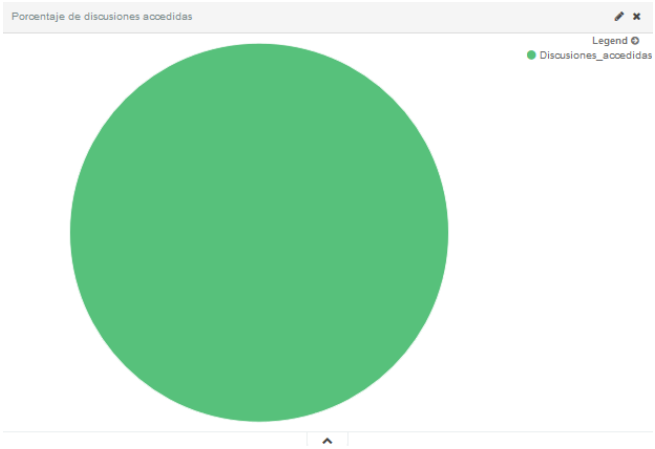


Figura 228. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE FOROS" – Visualización: Porcentaje de discusiones accedidas

En el panel **"ESTADÍSTICAS SOBRE USO DE MÓDULOS"** se analiza cómo los usuarios acceden a los módulos publicados en el curso. Para ello, se compone de las siguientes vistas, apreciables en la Figura 229.

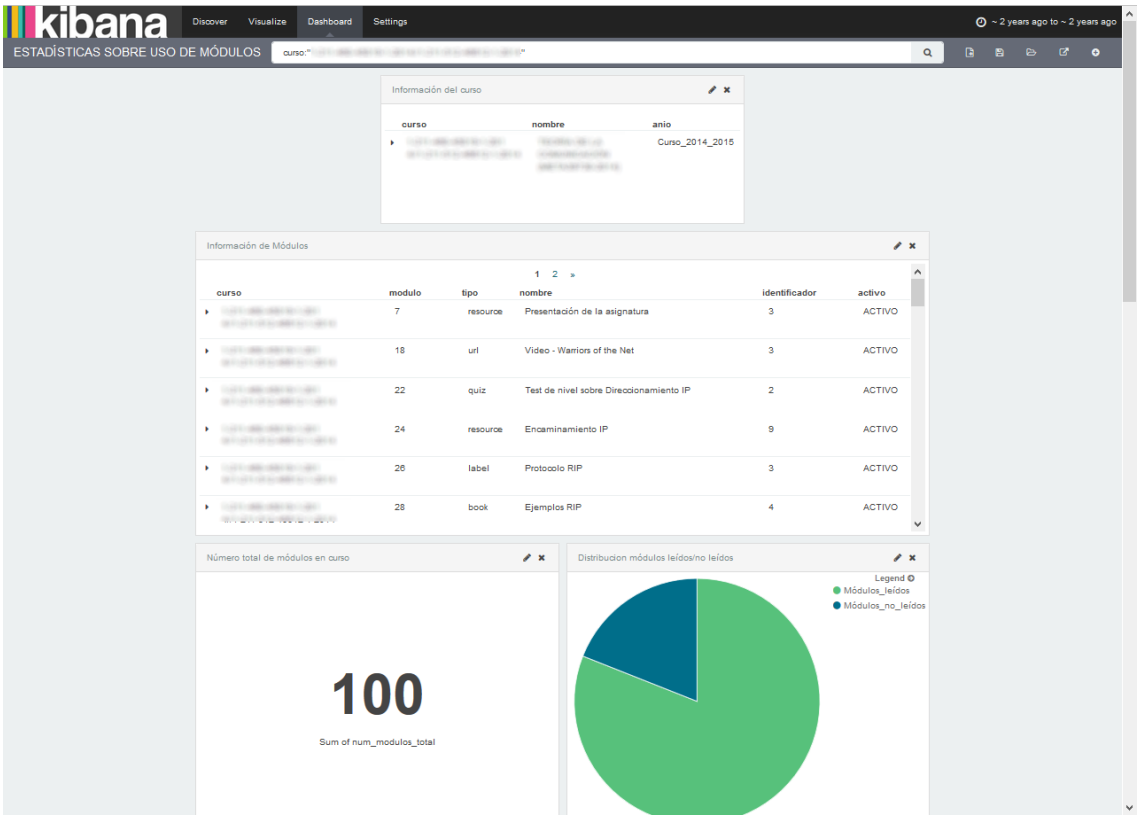


Figura 229. Vista previa del *dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS"

- Información del curso, similar a Figura 110.
- Información de módulos publicados en el curso (Figura 230).

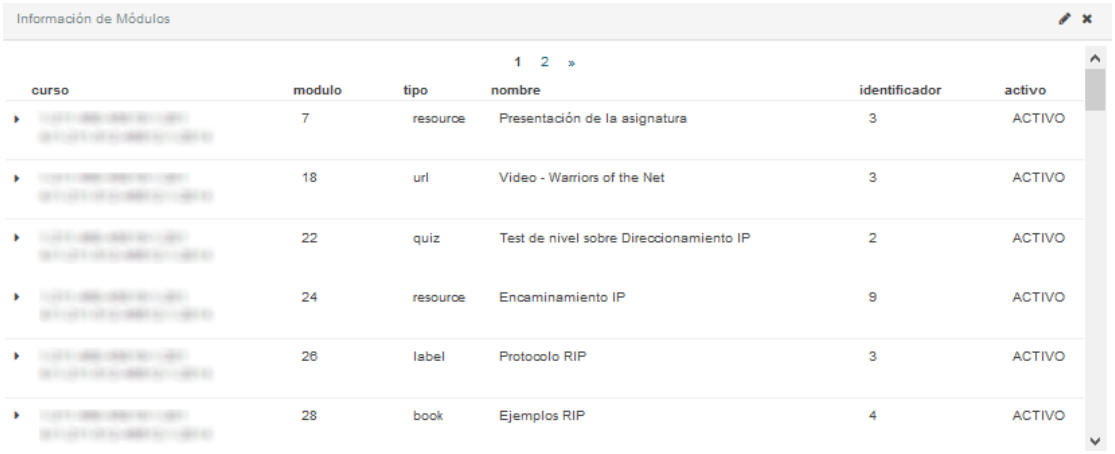


Figura 230. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS" – Visualización: Información de módulos

- Número total de módulos en el curso (Figura 231).

Figura 231. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS" – Visualización: Número total de módulos en curso

- Distribución de módulos leídos / no leídos (Figura 232).

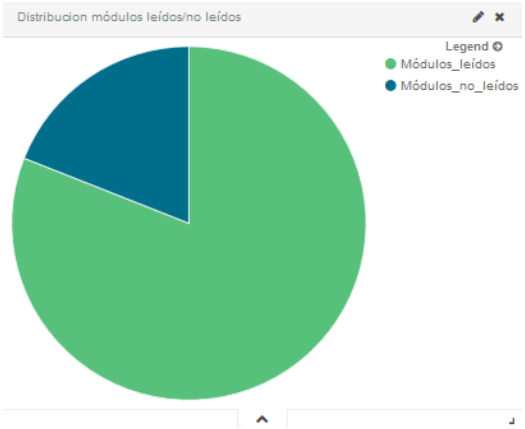


Figura 232. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS" – Visualización: Distribución módulos leídos / no leídos

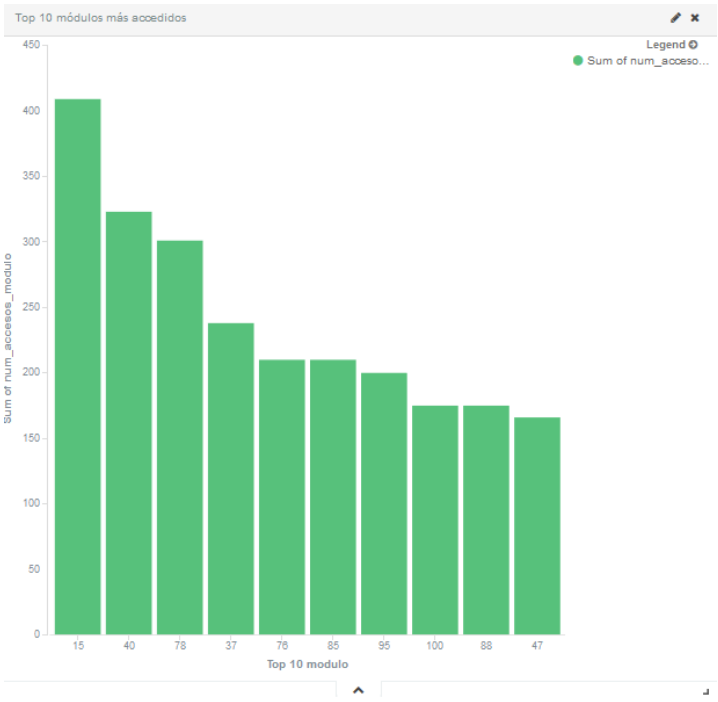


Figura 233. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS" – Visualización: Top 10 módulos más accedidos

- Top 10 módulos más accedidos (Figura 233).

- Top 10 de usuarios con más visitas a módulos y Top 5 de módulos más visitados por cada usuario (Figura 234).

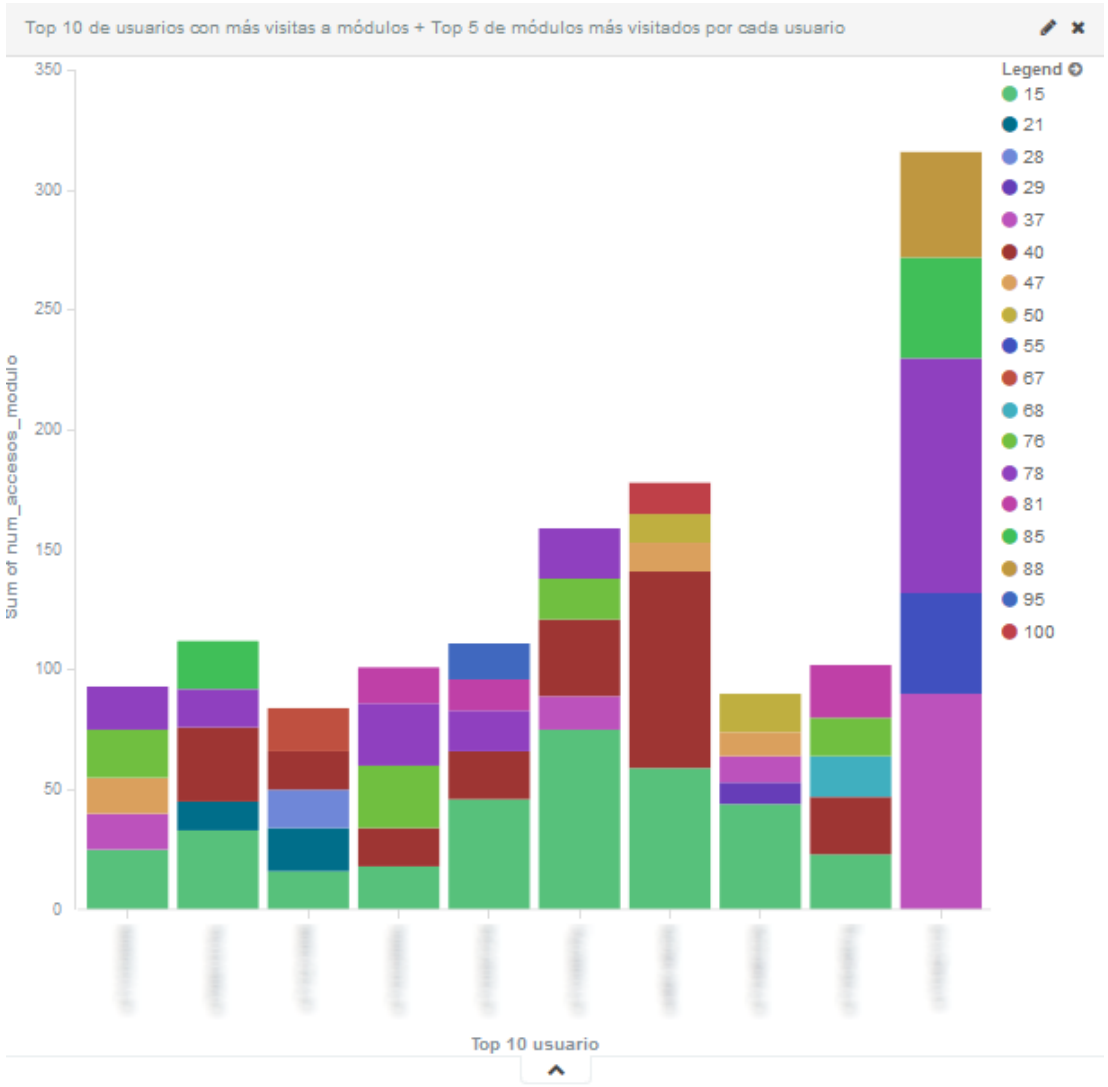


Figura 234. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS" – Visualización: Top 10 de usuarios con más visitas a módulos + Top 5 de módulos más visitas por cada usuario

- Top 10 módulos con mayor porcentaje de lectura (Figura 235).

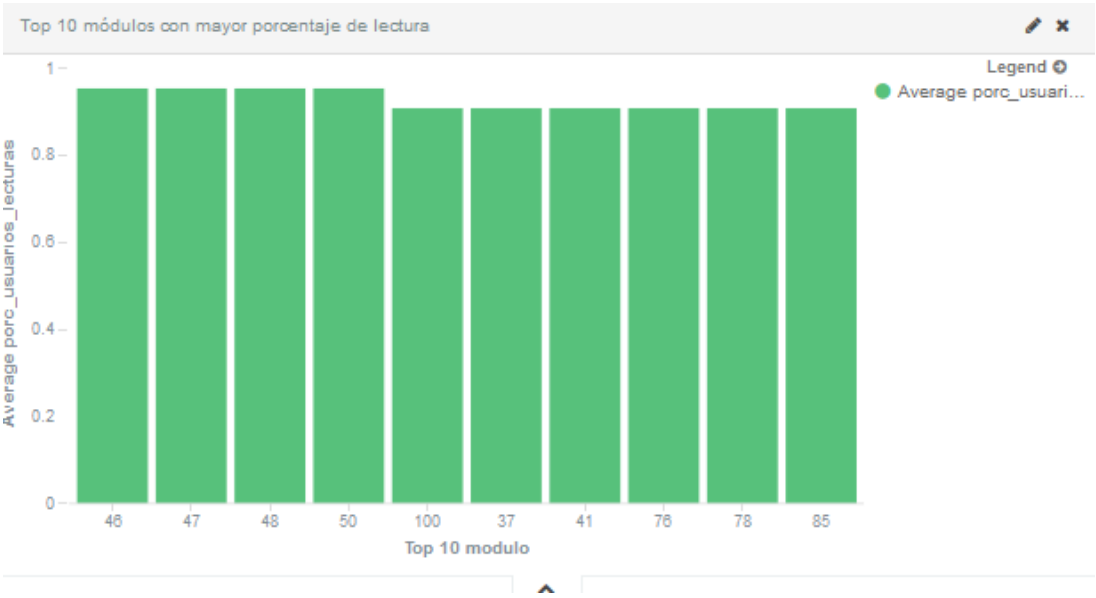


Figura 235. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS" – Visualización: Top 10 módulos con mayor porcentaje de lectura

- Correlación entre la nota final y el número de accesos a módulos, es decir, influencia del número de accesos a cada módulo sobre la nota final obtenida por los alumnos. Mismo gráfico que Figura 134.

- Representación gráfica de la relación anterior. Mismo gráfico que Figura 135.

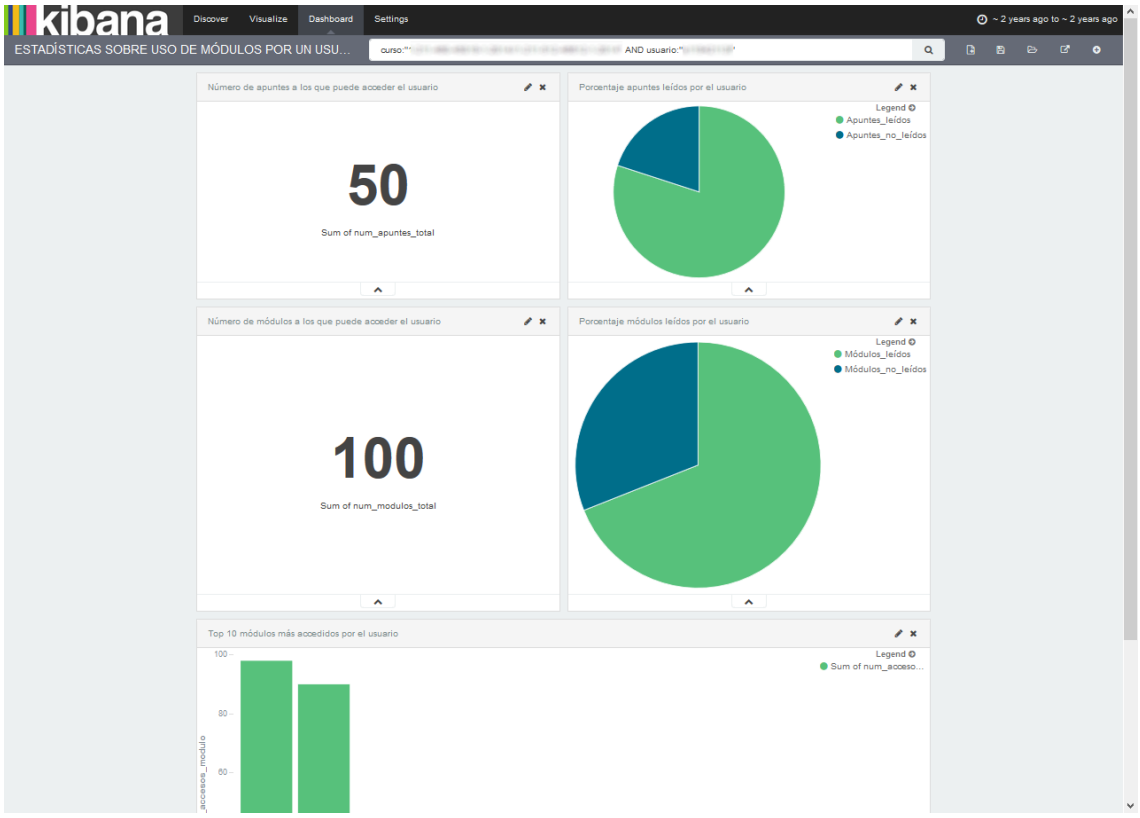


Figura 236. Vista previa del *dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS POR UN USUARIO"

En el panel **"ESTADÍSTICAS SOBRE USO DE MÓDULOS POR UN USUARIO"** se muestra el uso que un usuario en concreto ha realizado de los módulos publicados en un curso. Su vista previa se puede observar en la Figura 236. Se compone de las siguientes vistas.

- Número de apuntes a los que puede acceder el usuario (Figura 237).

Figura 237. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS POR UN USUARIO" – Visualización: Número de apuntes a los que puede acceder el usuario

- Porcentaje de apuntes leídos por el usuario (Figura 238).

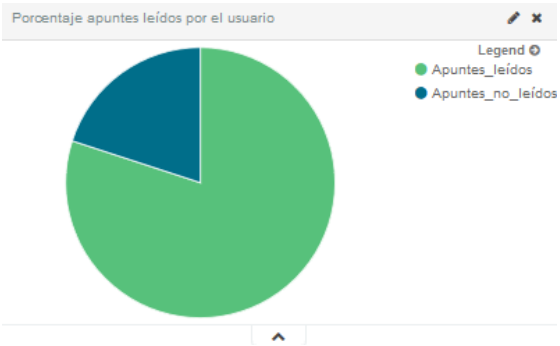


Figura 238. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS POR UN USUARIO" – Visualización: Porcentaje de apuntes leídos por el usuario

- Número de módulos a los que puede acceder el usuario (Figura 239).

Figura 239. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS POR UN USUARIO" – Visualización: Número de módulos a los que puede acceder el usuario

- Porcentaje de módulos leídos por el usuario (Figura 240).

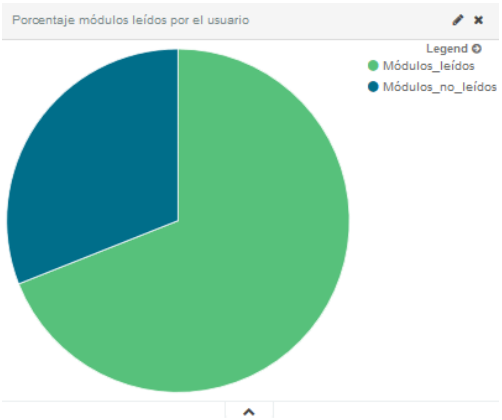


Figura 240. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS POR UN USUARIO" – Visualización: Porcentaje de módulos leídos por el usuario

- Top 10 módulos más accedidos por el usuario (Figura 241).

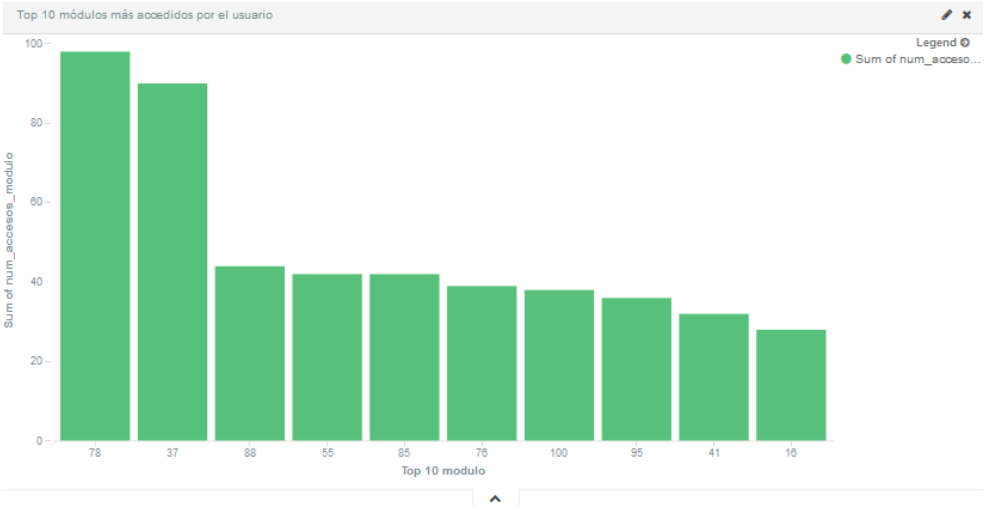


Figura 241. *Kibana*. *Dashboard* "ESTADÍSTICAS SOBRE USO DE MÓDULOS POR UN USUARIO" – Visualización: Top 10 módulo más accedidos por el usuario

En el panel **"REGLAS DE ASOCIACIÓN"** se muestran los resultados de la aplicación del algoritmo de reglas de asociación sobre el comportamiento de los alumnos del curso. Está compuesto por una vista que contiene estas reglas, como se puede apreciar en la Figura 242.

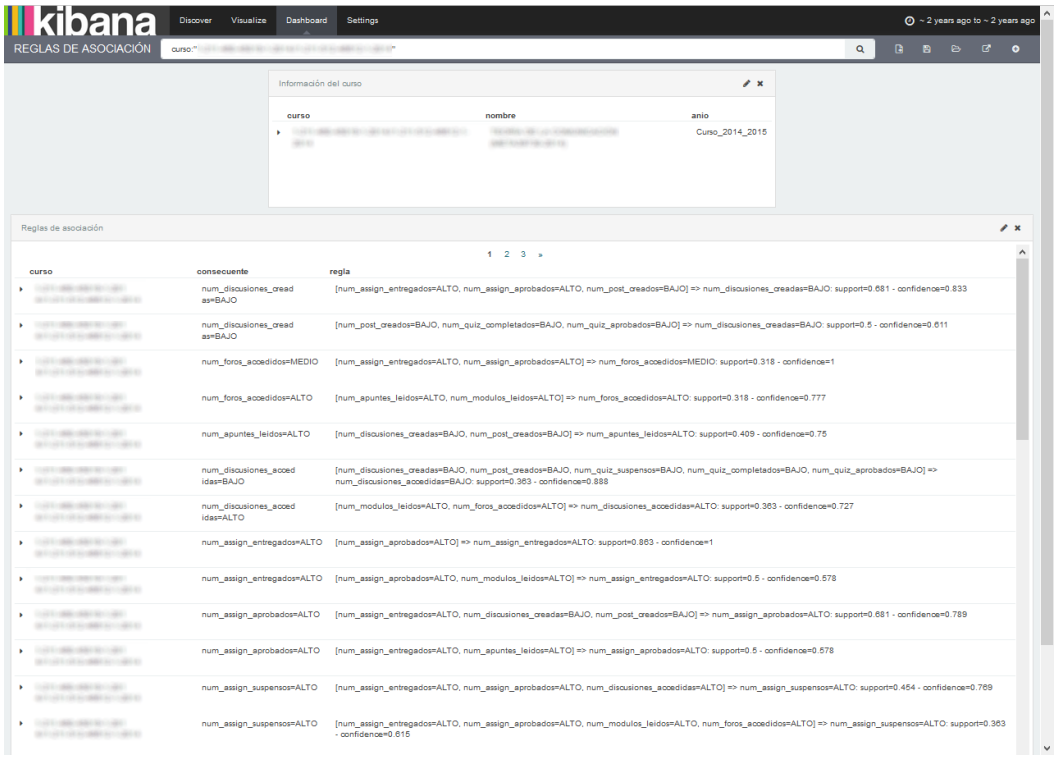


Figura 242. Vista previa del *dashboard* "REGLAS DE ASOCIACIÓN"

4.8. Automatización mediante *Oozie*

Apache Oozie es la herramienta que nos va a permitir automatizar la ejecución de los trabajos, tanto definir el orden en el que deben ejecutarse las sentencias que conducen a los resultados de los apartados 4.5, 4.6 y 4.7, como programar su ejecución diaria.

Para ello, en primer lugar, se introducirá el funcionamiento de *Oozie* así como la estrategia utilizada. En los siguientes tres apartados se enumerarán los elementos configurables de *Oozie* así como la forma en la que estos han sido utilizados. Finalmente, se presenta una representación gráfica de la solución utilizada para la automatización de nuestro trabajo.

4.8.1. ¿Qué es *Oozie*? Estrategia utilizada

Apache Oozie es una herramienta que nos va a permitir planificar la ejecución de trabajos en *Hadoop*, tal y como se presentó en el apartado 2.2.3.2.8. Principalmente, nos permite definir flujos de trabajo directos, los cuales pueden combinar distintos tipos de acciones.

Trabajar con *Oozie* es un poco más complicado que con el resto de herramientas, ya que la definición de los trabajos se debe realizar mediante ficheros en formato *hPDL*, parecido a *XML*. Como alternativa, podemos usar la interfaz que presenta *Hue*, en la cual, podemos ir dibujando los flujos de trabajo que queremos definir. En los próximos apartados se verá que se pueden definir tres tipos de elementos: *workflows*, *coordinators* y *bundles*. El problema encontrado está cuando necesitamos exportarlos a otra máquina, o por ejemplo, proporcionarlos como documentación adjunta en este trabajo. Se pueden conseguir los ficheros correspondientes, acudiendo al directorio de trabajo escogido para el flujo de trabajo, pero el mayor problema

nos lo encontramos cuando queremos realizar la operación contraria: dado un fichero con la definición, transferirlo a *Hue*. Dependiendo de versiones de *Hue*, existe una opción de importar un *workflow*, pero no siempre funciona correctamente, ya que según lo que contenga el flujo de trabajo, puede no ser capaz de leerlo. No se pueden importar ni *coordinators* ni *bundles*.

Además, en la práctica nos hemos encontrado con problemas a la hora de especificar toda la configuración necesaria para los *coordinators*. Así, finalmente la estrategia seguida para la construcción de los ficheros ha sido:

- Los *workflows* se dibujan mediante *Hue*, pero después se obtiene su fichero de definición para ser ejecutados manualmente.
- Los *coordinators* y los *bundles* se definen manualmente a partir de ficheros de ejemplos de la documentación de *Oozie*.

Desde *Hue* se puede lanzar la ejecución de cualquier elemento. Desde línea de comandos usaríamos la siguiente línea:

```
oozie job –oozie http://localhost:11000/oozie -config job.propierties
                                    –run
```

Al ejecutar desde línea de comandos se debe tener cuidado con el usuario que lanza la acción, ya que si queremos controlar el estado de la ejecución desde *Hue*, se necesita que sea lanzada por el usuario `hue`:

```
sudo –u hue oozie job –oozie http://localhost:11000/oozie -config
                        job.propierties –run
```

Siendo requisito que el fichero `job.properties` sea accesible a este usuario, en el sistema de ficheros de la MV, no en *HDFS*. En este fichero, se define la configuración del trabajo que queremos lanzar. Suele tener un aspecto de la forma:

```
nameNode=hdfs://sandbox.hortonworks.com:8020
jobTracker=sandbox.hortonworks.com:8050
queueName=default
parametro1=valor1
oozie.use.system.libpath=true
#En esta propiedad se debe poner la ruta en donde está guardado el
#workflow.xml en HDFS
oozie.wf.application.path=${nameNode}/user/myexample
```

En él se definen, principalmente, las variables que se van a usar en los *workflows* que ponga en ejecución. Las dos primeras (siempre presentes) permiten definir la conexión con el *NameNode* y el *JobTracker* para ejecutar los trabajos. Con respecto a estos dos parámetros hay que tener cuidado porque los ejemplos de este fichero encontrados en la *sandbox* los tienen mal configurados.

La propiedad `oozie.use.system.libpath=true` también suele estar presente en todos los ficheros `job.properties`. *Oozie* tiene asociada una *ShareLib* donde se almacenan

todas las librerías necesarias para la ejecución de los trabajos. Esta, por defecto está ubicada en *HDFS* en la ruta `/user/oozie/share/lib/`, donde mantiene ordenadas las librerías según la acción que las va a utilizar. Por ejemplo, nos podemos encontrar un subdirectorio denominado `hive`, en el que se guardan todas las librerías que se van a usar cuando se ejecuten acciones de tipo de *Hive*, por lo que si nos falta alguna, la podemos añadir aquí. Por defecto, la *ShareLib* viene instalada, pero en caso de no ser así, se puede encontrar en la ruta `/usr/hdp/{versión}/oozie/share/lib` del sistema de ficheros de la MV. *Oozie* solo usa esta librería cuando se lanzan acciones si la opción indicada anteriormente está a `true`. Más adelante, se indicará otra forma de añadir librerías a la ejecución de una acción.

Por último, en el fichero `job.properties` se debe indicar el elemento que se quiere ejecutar, pudiendo ser:

- Un *workflow*, indicado mediante la opción `oozie.wf.application.path` junto con la ruta al directorio en *HDFS* que contiene el fichero `workflow.xml` con la definición.
- Un *coordinator*, indicado mediante la opción `oozie.coord.application.path` junto con la ruta al directorio en *HDFS* que contiene el fichero `coordinator.xml` con la definición.
- Un *bundle*, indicado mediante la opción `oozie.bundle.application.path` junto con la ruta al directorio en *HDFS* que contiene el fichero `bundle.xml` con la definición.

Los nombres de los ficheros de definición de los elementos, deben ser los indicados anteriormente.

Centrándonos en nuestro trabajo, mediante un *workflow* de *Oozie* solo se pueden definir flujos de trabajos directos, pero nosotros tenemos un conjunto de relaciones entre acciones mucho más complicado, por lo que no podemos definir toda la ejecución mediante un único *workflow*. Así, la alternativa utilizada consistirá en definir pequeños *workflows* que realicen cosas sencillas y que se puedan definir como flujos directos y englobar cada uno de ellos en un *coordinator*, ya que con este segundo elemento podemos definir condiciones de ejecución del tipo: para ejecutarse B debe haberse completado A. Por último englobaremos todos los *coordinator* dentro de un *bundle* para poder gestionarlos conjuntamente.

Como tenemos dos conjuntos de operaciones: inicial e incremental, se han implementados dos aplicaciones, es decir, dos conjuntos de *workflows*, *coordinators* y *bundles*: uno para implementar la aplicación inicial de *BigData* y otro para la incremental. Existe una ligera diferencia entre ambas, pero en el fondo son muy parecidas.

En los siguientes apartados se detallarán estas cuestiones. Más información, se puede encontrar en [213].

4.8.2. *Workflows*

Un *workflow* es un flujo de trabajo, es decir, la concatenación de forma paralela o secuencial de acciones, en el que definimos las dependencias de ejecución de las acciones de que consta el trabajo. Los *workflows* de *Oozie* son directos y acíclicos, en definitiva, muy simples, ya que implica que una vez se acceda a una bifurcación, esta no puede juntarse con otra al menos que provengan del mismo origen. Esto nos limitó el trabajo, ya que nuestro flujo total de trabajo, es muy complejo y no se puede definir como un flujo acíclico directo. Por lo que no podemos usar un *workflow* para definir el total del trabajo.

Como se ha comentado anteriormente, la alternativa va a ser definir un *workflow* por cada uno de los resultados que se generen. Anteriormente, se detalló que en el trabajo se habían definido una serie de bloques de ejecución, que, principalmente, generaban resultados sobre una tabla *Hive* tomando como partida los resultados de otros bloques anteriores. Cada uno de estos bloques se va a representar mediante un *workflow*, un flujo de trabajo que permite obtener ese resultado en *Hive*. La mayor parte de estos flujos van a ser muy simples, van a consistir en la ejecución secuencial o paralela de un par de acciones.

De entre todas las acciones que nos permite usar *Oozie*, vamos a usar cuatro: *Sqoop*, *Hive*, *Java* y *Shell*, dada la implementación del trabajo detallada a lo largo de este capítulo. A continuación se detalla cómo se deben configurar cada una de estas acciones.

- **Acción *Sqoop***:
    - En el apartado `command` se debe indicar el comando *Sqoop* a ejecutar. Será exactamente el mismo comando que se pondría por línea de comandos, menos la palabra `sqoop` del principio.
    - En algunas documentaciones se indica que se debe especificar la ruta al fichero de configuración de *Hive*, en *HDFS,* denominado `hive-config.xml` en el apartado `Files` o `Job XML`. En nuestro caso, parece que no ha hecho falta, ya que funciona correctamente.
    - El conector a utilizar con la base de datos debe copiarse en el directorio *Sqoop* de la *ShareLib* de *Oozie*, de forma parecida a como se indicó en el apartado 4.2.4.
- **Acción *Hive***:
    - En el apartado `Script Name` indicaremos la ruta, en *HDFS*, en la que se encuentra el fichero donde se han incluido las sentencias *Hive* a ejecutar.
    - En los apartados `File` (traducido como archivo si la interfaz está en español) y `Job XML` se debe indicar la ruta, en *HDFS*, donde se encuentra el fichero `hive-config.xml`. Este fichero contiene información sobre la configuración *Hive* con la que se va a lanzar las consultas. No tiene un por qué, pero sería conveniente que fuera exactamente igual a la configuración *Hive* dada a la máquina, cuya configuración la podemos encontrar en `/etc/hive/{versión}/0/hive-site.xml`. El fichero debe llamarse `hive-config.xml`, no `hive-site.xml`, ya que si no aparecerán errores de permisos de lectura.
    - A mayores, se deben añadir las librerías de *Hive Hook* a la *ShareLib* de *Hive* en *Oozie*. Estas librerías las podemos encontrar en el directorio `/usr/hdp/{versión}/atlas/hook/hive/`. También debemos añadir el conector *Hadoop – ElasticSearch* en este mismo directorio de la *ShareLib.*
- **Acción *Shell***:
    - En el apartado `Exec` debemos especificar el comando a ejecutar. Opcionalmente, si se necesita, en el apartado `File` deberemos indicar la ruta, en *HDFS*, a un fichero de ejecución. En este trabajo, las acciones *Shell* ejecutarán un *script*, por lo que en `Exec` se indicará el nombre del fichero que contiene el *script*, y `File`,

la ruta al fichero, ya que para ejecutar un *script* en *Shell*, solo hay que indicar su nombre de fichero.

- **Acción *Java***:
    - En el apartado `main-class` le debemos indicar la clase principal a ejecutar.
    - No hace falta añadir más configuración si el programa *Java* ejecutable *JAR* se encuentra en una carpeta denominada `lib` en el mismo directorio que el fichero `workflow.xml`. Se supone que podemos especificar la ruta al fichero *JAR* en el apartado `Jar Name`, pero esto en la práctica ha dado muchos errores, por lo que dada la simplicidad de la carpeta `lib`, se ha optado por esta opción. Así cambiar el nombre del *JAR* no implica cambiar la definición del *workflow*.

En todos los casos, cuando se debe indicar una ruta a un fichero o similar se recomienda hacerlo de forma absoluta, por ejemplo, si el fichero está en la ruta, en *HDFS*

```
/user/examples/script.sql
```

Es recomendable indicarlo de la forma

```
${nameNode}/user/examples/script.sql
```

Además, si no se especifica ruta, se suele interpretar que se debe buscar el fichero en el área de trabajo. Mediante el panel de *workflow* se puede ver el progreso de las acciones y los posibles errores, así como a través de la interfaz de *Oozie*.

Cuando se lanza cualquiera de estas acciones, se hace como un trabajo *Oozie*, el cual consistirá en un trabajo *MapReduce*. Esto implica, que, por ejemplo, una acción *Hive* lanzada mediante su terminal cliente, genera un trabajo *MapReduce* (realmente Tez), pero al lanzarlo sobre *Oozie*, genera dos: uno para *Oozie* y otro para la propia acción *Hive*. El primer trabajo relativo a *Oozie* se denomina *Launcher*, y consiste en un trabajo *MapReduce* pero solo con parte *Map*. Este se encarga de preparar la acción correspondiente a ejecutar, por ejemplo, buscando las librerías necesarias y lanzando la acción en el servicio que corresponda.

Esta forma de ordenar la ejecución, en la práctica, puede dar lugar a un problema de bloqueo, conocido como "*deadlock*". Si recordamos la forma de funcionamiento de *YARN* (apartado 2.2.3.2), en primer lugar el *Launcher*, instanciará un *Application Master* para su aplicación, lo cual requerirá de un *container* de recursos. Desde este, se negocia por, al menos un *container* de ejecución del *Launcher*. Desde este *container*, se lanza el *Application Master* de la acción correspondiente, desde el que a su vez se lanzan los *containers* de ejecución de la acción. Hasta que este último conjunto de *containers* no termine su ejecución, no se van a liberar los recursos de los tres *containers* anteriores.

De esta forma, si ejecutamos varias acciones en paralelo, pueden producirse problemas de bloqueo. Por ejemplo, supongamos que en nuestra infraestructura *Hadoop* tenemos recursos (memoria *RAM*) para lanzar cuatro *containers*. *YARN* funciona de tal forma que si existen varias tareas a ejecutar a la vez, aquellas que no encuentren recursos van a esperar a que estos sean liberados. Supongamos que tenemos dos acciones *Hive* en paralelo. Cada una de ellas cogerá un primer *container* para el *Application Master* del *Launcher* y un segundo *container* para la ejecución del *Launcher*, es decir, ya se han ocupado los cuatro *containers*, por lo que si hay

más aplicaciones en ejecución, deberán esperar. Esta situación genera un bloqueo, ya que como no existen *containers* libres, no se podrán lanzar los *Application Master* y lo que sigue de las acciones *Hive* correspondientes. Estas no se lanzarán nunca ya que no se van a liberar recursos. Para que se liberen los recursos de los *Launcher* deben finalizar sus acciones *Hive*, pero estas nunca van a comenzar porque los *Launcher* han acabado con la capacidad del *cluster*.

La solución a este problema es trabajar con distintas colas, por ejemplo, una cola en la que se lancen los *Launcher* y otra para las acciones, y repartir entre ellas la capacidad del *cluster*, por ejemplo 50:50. De esta forma, se lanzarán *Launcher* hasta alcanzar la mitad de la capacidad, dejando la segunda mitad para la ejecución de sus acciones contenidas.

En este trabajo se han definido dos colas, siguiendo la configuración de la Figura 243.

- *default*: cola por defecto donde se llevan los trabajos si no se indica nada.
- *launcher*: cola donde se van a enviar los *Launchers* de las acciones.

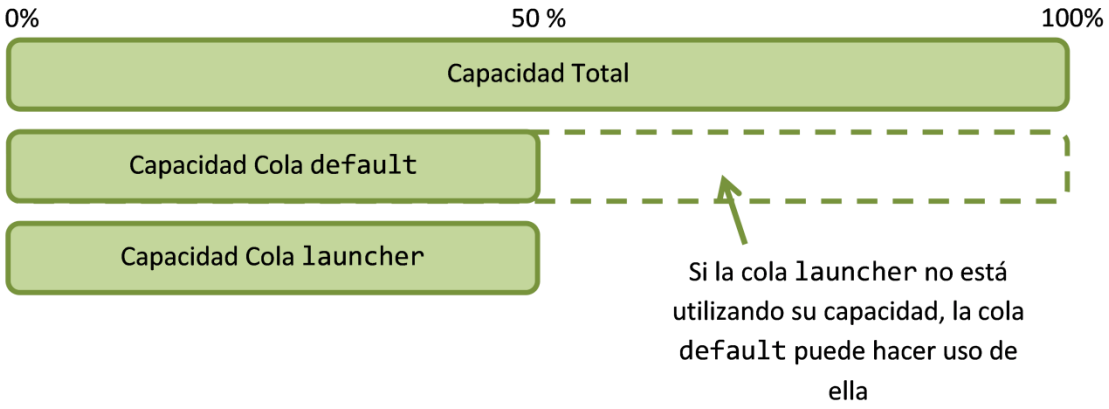


Figura 243. Configuración de colas *YARN*

Cuando definimos un *workflow* para cada acción, se puede configurar a qué cola queremos enviar tanto el *Launcher* como la acción contenida. En nuestro caso, indicaremos que el *Launcher* debe ir a la cola *launcher*, mediante la propiedad de ejecución: `oozie.launcher.mapred.job.queue.name`. Las acciones contenidas se envían directamente a la cola *default*, ya que no se especificará ninguna propiedad al respecto.

En concreto, se ha aplicado la siguiente gestión de colas:

- Para las acciones *Sqoop* y *Hive*, su *Launcher* se envía a la cola *launcher* y la acción se ejecuta a través de *default*.
- Para las acciones *Shell*, el *Launcher* se ejecuta a través de *default* ya que se ha observado que es dentro del propio *Launcher* donde se ejecuta la acción.
- Las acciones *Java* también se ejecutan dentro del propio *Launcher*, pero en nuestro caso, como los programas *Java* llevan contenidas conexiones con la base de datos *Hive*, genera *subacciones Hive* y para evitar un problema de bloqueo similar, también se configura el *Launcher* a través de la cola *launcher*, mientras que las acciones contenidas en el programa *Java* se ejecutarán a través de la cola *default*.

Por último, con respecto a las acciones *Java*, estas van a requerir de otras librerías a mayores del propio programa *Java*. Se plantearon diversas alternativas para hacerlas referencia. Al final se optó por lo que se explica a continuación.

El programa *JAR* está compilado de tal forma que se supone que el ambiente de trabajo ya habrá cargado las librerías necesarias, es decir, se genera sin ningún fichero de manifiesto que indique dónde buscar las librerías necesarias. Para que el ambiente de trabajo cargue estas librerías directamente tenemos tres opciones:

- Cuando *Oozie* lanza una acción *Java*, considera como entorno de trabajo todas las librerías que se encuentren en el directorio `lib` del directorio del *workflow*, donde se ha ubicado el *JAR* ejecutable. Podemos también incluir las librerías requeridas. En la práctica no se ha usado esta opción, ya que los tres programa *Java* construidos requieren de prácticamente las mismas librerías y se tendría información triplicada en *HDFS*.

- Cuando *Oozie* lanza una acción, carga todas las librerías ubicadas en su *ShareLib*. En el caso de *Java*, estas deberían estar ubicadas en el directorio `oozie` dentro de este *ShareLib*. Esto genera un problema, y es que estas librerías pasaran a ser dependencias de todas las acciones, sean del tipo que sea, porque todas cargan las dependencias de este directorio y luego del propio. No provoca un error de ejecución, sino un problema de carga de datos, ya que cuando una acción debe ejecutarse, tras haber elegido el nodo, la parte del *ShareLib* que se vaya a utilizar se copia a ese nodo. Estaríamos transfiriendo grandes cantidades de datos sin utilidad (las librerías necesarias alcanzan los 100MB).

- Se puede indicar en la definición de la acción, a mayores de las dependencias en el directorio `lib` y en la *ShareLib* de *Oozie*, otros directorios, mediante la propiedad `oozie.launcher.oozie.libpath`. Esta será la opción empleada ya que nos permite que solo las tres acciones *Java* para la ejecución de los tres algoritmos de *Machine Learning* usen estas librerías como dependencias así como solo tener una copia de las librerías.

Con respecto a la configuración dada en la generación de cada *workflow*, remitimos al código fuente. A pesar de haber múltiples acciones de cada tipo, cada tipo de acción se ha configurado siguiendo un patrón, es decir, todas las acciones *Hive*, se han configurado igual, siguiendo las cuestiones planteadas previamente, y solo se ha cambiado los nombres de las rutas a los directorios o ficheros necesarios. Lo mismo para las acciones *Sqoop*, *Shell* y *Java*. Así, se considera no ser demasiado complicado de entender a partir del código fuente y no se plantean más detalles en este documento.

Se puede encontrar más información sobre la definición de *workflows* en [214].

4.8.3. *Coordinators*

Los *coordinators* nos permiten definir y controlar calendarios de ejecución de los flujos de trabajo definidos. En nuestro caso, además nos permitirán dibujar un diagrama de flujo complejo que contenga los distintos *workflows* anteriormente definidos y controle su ejecución en el orden correcto.

Este trabajo se ha estructurado tal que cada *workflow* está contenido en un *coordinator* del mismo nombre, el nombre que caracteriza al bloque de ejecución que contienen. Es decir, el bloque `Cursos`, que se encarga de elaborar las tablas *Hive* relativas a los cursos de *Moodle*, se

ejecuta por medio de un *coordinator* denominado `Cursos`, que a su vez controla un *workflow* también denominado `Cursos`, siendo este último, donde se definen los *scripts Hive* que contienen las sentencias para la generación de las tablas.

Entonces, para definir cada *coordinator* debemos tener en cuenta dos criterios:

- Temporización: calendario, fechas, horas de ejecución, cada cuanto tiempo repetir la ejecución, etc.
- Control del orden de ejecución: definir de qué otros bloques/*coordinators* depende la ejecución de este bloque, es decir, si el bloque `Matriculaciones` necesita información del bloque `Cursos`, se debe definir la dependencia para que uno no empiece a ejecutarse hasta que haya terminado el otro.

Se pueden encontrar muchas opciones de definición de *coordinator* [215], pero aquí se va a presentar lo necesario para la realización de este trabajo. A pesar de que se han definido muchos *coordinators*, todos ellos, siguen una estructura similar:

```
<coordinator-app name="Cursos" frequency="${coord:days(1)}"
    start="2016-03-04T00:00Z" end="2017-01-23T15:16Z" timezone="Europe/Madrid"
xmlns="uri:oozie:coordinator:0.2">
       <controls>
              <timeout>${coord:days(7)}</timeout>
              <concurrency>1</concurrency>
              <execution>LIFO</execution>
              <throttle>12</throttle>
       </controls>
       <datasets>
              <include>
               hdfs://sandbox.hortonworks.com:8020/user/MoodleBigData/dataset/
               dataset-incremental.xml
              </include>
       </datasets>
       <input-events>
              <data-in name="input1" dataset="UltimoDia">
                     <instance>${coord:current(0)}</instance>
              </data-in>
              <data-in name="input2" dataset="DatosOrigen">
                     <instance>${coord:current(0)}</instance>
              </data-in></input-events>
       <action>
              <workflow>
                     <app-path>
                      hdfs://sandbox.hortonworks.com:8020/user/MoodleBigData/apps/
                      incremental/Cursos
                     </app-path>
              </workflow>
       </action>
</coordinator-app>
```

En cualquiera de las definiciones podemos distinguir cuatro partes, detalladas a continuación.

Temporización del coordinator, incluida en la cabecera de definición. Mediante las propiedades que a continuación se enumeran, se fija el calendario de ejecución de las acciones que contenga el *coordinator*:

- `frequency`: variable de repetición, por ejemplo cada día.

- `start`: fecha de inicio de ejecución.
- `end`: fecha de fin de ejecución.
- `timezone`: huso horario de ejecución.

Con respecto a estos valores hay varias cuestiones. A pesar de que introduzcamos una zona horaria, las fechas se tomarán en UTC/GMT. Es decir, si indicamos como fecha de inicio 2016-03-03T14:25Z lo estará tomando en hora UTC, por lo que, suponiendo horario de invierno, en el inicio será a las 15:25 hora española. Las fechas deben especificarse en UTC/GMT, así deberíamos poner 2016-03-03T13:25Z+02:00 si queremos empiece a las 14:35 hora española. Exactamente igual que se modificó el huso horario de la MV y de *Hue* podríamos modificar la configuración de *Oozie* para ponerle hora española. Pero esto provoca un problema: cuando se configura un *coordinator* hay que especificar fecha/hora de inicio y fin de la planificación o de instantes concretos de ejecución del *workflow*, por ejemplo, de la forma 2009-02-01T00:00Z (si se selecciona la hora mediante *Hue* se realiza seleccionando el día de un calendario y la hora de un desplegable). Si configuramos la hora de *Oozie* para ser la española, esta será GMT+1 (horario de invierno) o GMT+2 (horario de verano). Esto implica, que en los *coordinators* la hora se deba expresar de la forma: 2009-02-01T00:00+0200. Si el *coordinator* se define manualmente, creando el fichero, no hay ningún problema porque el usuario puede definir en este formato las fechas/horas, pero cuando se definen mediante *Hue*, no las guarda en este formato, ya que este es un *bug* del programa o simplemente no se ha configurado para que funcione así, ya que se puede ver la presencia de un mensaje de advertencia que dice que solo se acepta tiempo UTC/GMT. Esto implica que las horas que queramos poner en hora española deberemos transformarlas a hora a GMT. Todo esto quiere decir, si estamos en horario de invierno, la hora española peninsular es GMT+1, si queremos que el programa se ejecute un día en concreto a las 16:00 horas, deberemos configurar un *coordinator* con hora 15:00, ya que cuando en GMT+1 son las 16:00 horas, en GMT son las 15:00.

Para evitar este problema, se ha optado por dejar la hora de *Oozie* en UTC y manejar la diferencia de hora con la española en las fechas de inicio y fin, restando una o dos horas según el horario de invierno o verano. Por ello, todos los *coordinator* tendrán como hora de inicio las 00:00 sabiendo que realmente será a la 01:00 en invierno y 02:00 en verano, para dejar un margen de tiempo en el cambio de día.

Los *coordinators* siempre deben ser síncronos, algo necesario en nuestra aplicación incremental, pero no en la inicial. Por ello, la aplicación inicial se ha definido de tal forma que solo se ejecute una instancia. Para ello, se ha fijado una frecuencia de 12 meses y unas fechas de inicio y fin tal que entre ambas haya menos de 12 meses, así solo habrá una ejecución. Las fechas de la aplicación incremental deberán ser fijadas en función de cuándo se vaya a utilizar la aplicación. Se recomienda que la fecha de inicio no sea anterior al día actual, para evitar que *Oozie* trate de recuperar acciones pasadas; nunca lo hará por la forma de programar los inicios, pero estas permanecerán en espera y esto puede sobrecargar el sistema.

El **control de la ejecución** se determina mediante las siguientes propiedades:

- `timeout`: tiempo máximo que una ejecución del *coordinator* puede estar esperando a comenzar su ejecución, es decir, en estado *WAITING*. Si no queremos especificar tiempo máximo, -1.
- `concurrency`: número máximo de instancias del *coordinator* que se pueden ejecutar (estado *RUNNING*) simultáneamente.

- `execution`: orden de ejecución si existen múltiples instancias del *coordinator*. Posibilidades: *FIFO*, *LIFO*, *LAST_ONLY* (solo la última que se creó, es decir, aquella con mayor fecha de inicio de ejecución), *NONE* (se descartan todas las materializaciones cuya fecha de inicio de ejecución es anterior a la fecha actual).

- `throttle`: número máximo de instancias del *coordinator* que pueden estar en estado *WAITING*.

En nuestro caso, para cada aplicación se han fijado los valores de la Tabla 144.

	aplicación inicial	aplicación incremental	aplicación incremental bloque DatosOrigen
timeout	-	7 días	22 horas
concurrency	1	1	1
execution	LAST_ONLY	LIFO	LAST_ONLY
throttle	1	12	1

Tabla 144. Valores de ejecución de los *coordinators* definidos

En la aplicación inicial los valores no tienen mucho interés ya que está preparado para ejecutarse solo una vez. En el caso de la aplicación incremental se han fijado así por motivos de ejecución en la práctica. El primer bloque que se ejecuta es `DatosOrigen`, encargado de traer la información de *Moodle* a *BigData*. Si este no se ejecuta, los demás tampoco. Supongamos que tenemos una aplicación incremental que empieza a ejecutarse un lunes, se retrasa demasiado y cuando llega la nueva ejecución de `DatosOrigen` del martes, todavía no ha acabado la del lunes, esta se queda en espera a que termine la del lunes. Si se retrasa tanto que llega la ejecución del miércoles, se descarta la del martes y cuando termine la del lunes empezará la del miércoles. Se ha perdido la ejecución de un día, pero se intenta reenganchar al día actual para no llevar retraso de un día en la ejecución.

Con el resto de bloques no podemos hacer lo mismo, ya que para ejecutarse, la dependencia es por fecha, es decir, el bloque de `Matriculaciones` depende de la finalización del de `Cursos`, pero para la misma fecha. Cada *coordinator* se lanza con una fecha, la fecha de inicio de espera (aunque la de ejecución puede ser otro día si tiene que esperar mucho). Así, la ejecución del bloque de `Matriculaciones` del lunes dependerá del término del bloque `Cursos` del mismo lunes. Si se retrasa la ejecución y se pasa al martes, si aplicamos una ejecución *LAST_ONLY*, se descarta la ejecución del lunes de `Matriculaciones` y salta la del martes, la cual esperará a que se ejecute el bloque `Cursos` del martes. Esto genera una situación de bloqueo, ya que el flujo del lunes quedará cortado, porque el bloque de `Matriculaciones` del lunes se ha descartado. Se ha pasado la ejecución al bloque del martes, el cual no se ejecutará, ya que el primer bloque, `DatosOrigen`, no se va a ejecutar hasta que termine el flujo del lunes.

Teniendo esto en cuenta, para el resto de bloques, se ha puesto otra forma de ejecución y un `throttle` de 12, para que puedan existir hasta 12 instancias en espera; si no, estaríamos en el mismo caso, cuando salte la del martes, se descartarán instancias para no superar el número máximo de instancias en espera. Con esta nueva configuración, cuando llegue el martes, tanto el bloque del lunes como del martes de `Matriculaciones` estarán en espera, y se podrá continuar con el flujo del lunes. La instancia del martes se ejecutará o no dependiendo de si se ejecuta `DatosOrigen` del martes, por ejemplo, bajo los supuestos anteriores, esta instancia se descarta, porque se pasa a la del miércoles, por lo que todo el flujo del martes quedará en espera. Aquí entra en juego `timeout`, fijamos que pasados 7 días, aquellos bloques que no han entrado en juego, se descartarán. Es un margen muy grande, lo deseado es que todas estas situaciones no ocurran, porque si no, el planteamiento de ejecución diaria no sirve

para el análisis, ya que este lleva más de un día en ser realizado. Si estas situaciones se plantean de forma regular, se debe reformular la estrategia.

Pero la parte más importante de nuestros *coordinators* no está en la temporización, si no en su **ejecución condicional**. Es decir, definir que el contenido del *coordinator*, por ejemplo, `Matriculaciones`, no puede ejecutarse hasta que haya terminado otro, por ejemplo `Cursos`. Para ello, se deben definir *DataSet* y Eventos de Entrada.

Un *DataSet* es una referencia a un dato que debe existir para el inicio efectivo del *coordinator*. Es decir, el *coordinator* saltará en ejecución en la fecha que hayamos programado, pero no entrará realmente en ejecución hasta que se haya generado ese dato. En *Oozie* esto está pensado para que ese dato forme parte de los datos de entrada de la ejecución. En nuestro caso simplemente va a ser un *flag* que avise que los datos están preparados. Será por medio de un evento de entrada por donde definamos las dependencias del *coordinator*. Por ejemplo, en el extracto anterior, la ejecución del *coordinator* `Cursos` está condicionada a la existencia de los *DataSet* `UltimoDia` y `DatosOrigen`. Los *DataSet* se pueden definir en el mismo *coordinator*, o en un fichero aparte, algo interesante cuando varios *coordinators* comparten el mismo *DataSet*. Por ejemplo, podríamos definir esos dos *DataSet* de la siguiente forma:

```
<datasets>
       <dataset name="DatosOrigen" frequency="${coord:minutes(1)}"
             initial-instance="2016-01-20T23:00Z" timezone="Europe/Madrid">
              <uri-template>
                     ${nameNode}/user/MoodleBigData/flags/
                     DatosOrigen/${YEAR}-${MONTH}-${DAY}
              </uri-template>
       </dataset>
       <dataset name="UltimoDia" frequency="${coord:minutes(1)}"
             initial-instance="2016-01-20T23:00Z" timezone="Europe/Madrid">
              <uri-template>
                      ${nameNode}/user/MoodleBigData/flags/
                      UltimoDia/${YEAR}-${MONTH}-${DAY}
              </uri-template>
       </dataset>
</dataset>
```

Los *DataSet* también son elementos síncronos, pero en este trabajo no importa demasiado. Cuando se define un *DataSet* se debe indicar la ruta en *HDFS* donde se generarán los datos. Por defecto, si no se especifica nada, como es este caso, el *flag* de completitud será un fichero denominado `_SUCCESS` en el directorio indicado en `uri-template`. Se pueden consultar más opciones en [216]. El atributo `frecuency` permite definir la frecuencia de regeneración de la información en ese directorio. En este caso se ha definido un minuto, porque no es un parámetro que importe demasiado. Cuando se lance, por fecha, el *coordinator* `Cursos`, supongamos en fecha 2016-03-03, va a buscar la existencia de los dos siguientes ficheros

`${nameNode}/user/MoodleBigData/flags/DatosOrigen/2016-03-03/_SUCCESS`

`${nameNode}/user/MoodleBigData/flags/UltimoDia/2016-03-03/_SUCCESS`

para comenzar su ejecución. Esto es debido a que se ha indicado que tome la instancia actual del *DataSet* (`${coord:current(0)}`). Si quisiéramos depender de instancias anteriores o posteriores, se debería configurar correctamente el campo `frequency`. Realmente la instancia actual (`current(0)`) contiene hasta el minuto exacto en que se lanzó el `coordinator`, pero en nuestro caso solo nos interesa la parte de fecha. La instancia anterior (`current(-1)`) se referiría al minuto anterior al de lanzamiento, si quisiéramos coger la instancia

de justo el día anterior, o bien quitamos 1440 minutos o configuramos la frecuencia del *DataSet* en un día y cogemos justo la instancia anterior. Viene conveniente detallado en la documentación de *Oozie*. Todos estos conceptos de temporales son bastante liosos y por ello se ha tratado de simplificar la configuración, aunque exista otra forma de realizarlo más elegantemente.

Obviamente para que esto funcione, se necesita generar el fichero _SUCCESS, por ello, la última etapa de cada *workflow* será la generación de ese fichero [217], en el directorio que le corresponda al bloque, mediante un *script* en *Shell* actuando sobre el sistema de ficheros de *HDFS*.

En el caso inicial, es más simple, ya que al solo existir una ejecución, la definición de los *DataSet* no lleva incluida la fecha del *coordinator*:

```
<datasets>
       <dataset name="DatosOrigen" frequency="${coord:minutes(1)}"
             initial-instance="2016-01-20T23:00Z" timezone="Europe/Madrid">
              <uri-template>
                     ${nameNode}/user/MoodleBigData/flags/DatosOrigen
              </uri-template>
       </dataset>
       <dataset name="UltimoDia" frequency="${coord:minutes(1)}"
             initial-instance="2016-01-20T23:00Z" timezone="Europe/Madrid">
              <uri-template>
                     ${nameNode}/user/MoodleBigData/flags/UltimoDia
              </uri-template>
       </dataset>
</dataset>
```

Por otro lado, para controlar que no se empiece a ejecutar un flujo mientras haya otro en ejecución, todos los bloques confluyen en uno denominado `Finalizado`, que depende de que todos ellos hayan finalizado su ejecución. Este se encarga de borrar los *flags* que todos ellos han generado, ya que llegados a este punto no tienen utilidad, y generar un *flag* de finalizado, para que pueda saltar el siguiente flujo. Es decir, la ejecución del bloque `DatosOrigen` que inicia un flujo, depende de que un flujo anterior haya generado este *flag*, además se encarga de borrarlo para que indicar que existe otro flujo en ejecución. En la aplicación inicial no existe esta dependencia, por cuestiones obvias. En la Figura 244 se trata de explicar gráficamente este comportamiento de forma genérica.

El *coordinator* de `DatosOrigen` incremental, también se encarga de guardar la fecha actual en el momento de iniciar el *coordinator* para que el resto de bloques la tomen para guardar los *flags* en el directorio con fecha correcta. Existe un margen de error: dado que los *coordinators* no inician a las 00:00 si no con un margen entre una y dos horas (según tipo de horario), si la ejecución de `DatosOrigen` se retrasa, por la ejecución tardía del flujo anterior, puede que `DatosOrigen` llegue a ejecutarse ya en el siguiente día, y todavía no se haya descartado esa instancia porque la siguiente no ha llegado. Así, en la práctica se puede fijar un `timeout` de 22 horas, para que una ejecución de `DatosOrigen` no intente ejecutarse en otro día (en el peor de los casos).

Para finalizar, en el *coordinator*, se debe indicar la **acción a ejecutar**, es decir, la ruta en *HDFS* donde se encuentra la definición del *workflow* que gestiona.

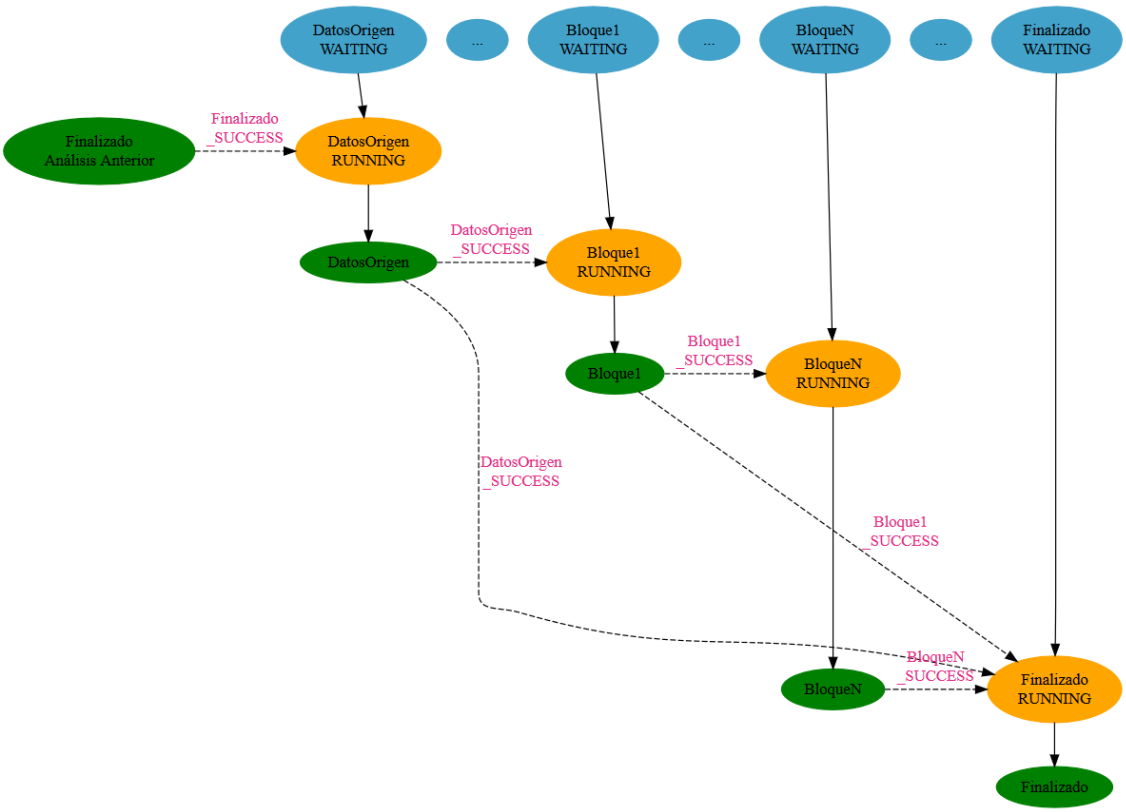


Figura 244. Ejemplo de ejecución condicionada de *coordinators*

4.8.4. *Bundles*

Un *bundle* es simplemente un paquete en el que agrupar varios *coordinator* o *workflows* y poder gestionar conjuntamente su ejecución. Por ejemplo, en nuestro caso, tenemos una lista de hasta 87 bloques cada uno de ellos representando un *coordinator*. Cuando queramos poner en marcha la aplicación, no se debe lanzar uno a uno los *coordinator*, simplemente se agrupan bajo un *bundle*, y lanza este último. Se puede encontrar más configuración en [218].

4.8.5. Representación gráfica de la solución

Para finalizar la explicación de *Oozie*, se presentan los gráficos que ilustran la solución planteada, en concreto, la dependencia entre bloques. Dada su enorme complejidad no es posible mostrar todas las dependencias simultáneamente, por lo que se mostrarán por conjuntos de bloques. Tampoco se ha incluido la confluencia de todos los bloques en el bloque de `Finalizado`.

En la Figura 245, se plantea el gráfico general a un nivel poco detallado, donde los bloques de ejecución se han agrupado en una de ocho categorías:

- *DatosOrigen*: bloque `DatosOrigen` que se encarga de transferir la información desde la base de datos origen y preprocesarla.
- *UtimoDia*: bloque `UltimoDia` para fijar la fecha del análisis, entre otros.

- *Información Moodle*: bloques relacionados con la información directa de *Moodle* transformada en tablas historial.
- *Estadísticas de acceso*: bloques encargados de generar las estadísticas de acceso.
- *Calificaciones*: bloques encargados de obtener información sobre *Assign*, *Quiz* y calificaciones finales del curso.
- *Correlaciones*: bloques encargados de calcular las correlaciones que nos dan información sobre la influencia entre conceptos.
- *Machine Learning*: bloques encargados de aplicar los algoritmos de *Machine Learning*.
- *Presentación*: bloques encargados de transferir resultados a *ElasticSearch* para su representación mediante *Kibana*.

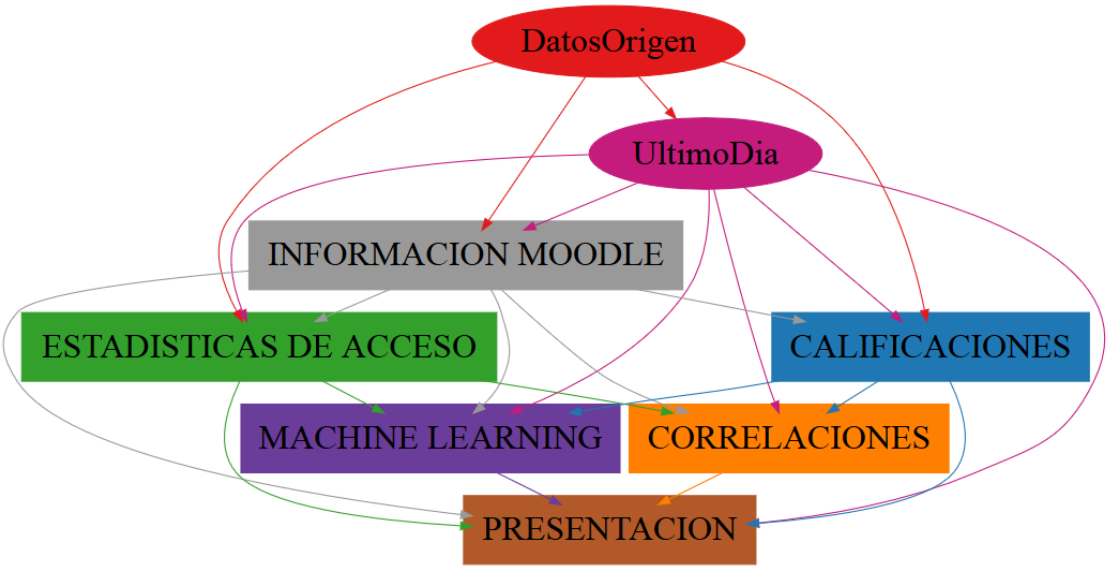


Figura 245. Representación gráfica de la solución a nivel no detallado

Tomando como referencia la Figura 245, se detallará el contenido de cada conjunto de bloques, así como las dependencias con otros conjuntos. En todos los casos, la dependencia entre *DatosOrigen* y *UltimoDia* se debe entender como un punto de partida, es decir, el bloque `UltimoDia` no necesita de la información de `DatosOrigen` para ejecutarse, pero se ha especificado esa dependencia para fijar un inicio de ejecución de las sentencias *Hive* una vez haya terminado toda la transferencia desde la base de datos *Moodle*.

El conjunto ***Información Moodle*** agrupa los siguientes bloques, relativos a la obtención de las tablas de historial histórico de la información de *Moodle*, enumeradas en la sección 4.6.2:

- `Cursos`.
- `Matriculaciones`.
- `Modulos`.
- `Foros`.
- `Discusiones`.

- `Grupos`.
- `Assign`.
- `Quiz`.

En la Figura 246 se presenta el contenido de este conjunto en el caso de la carga inicial: cómo obtener cada bloque así como las dependencias entre ellos. La carga incremental se representa en la Figura 247.

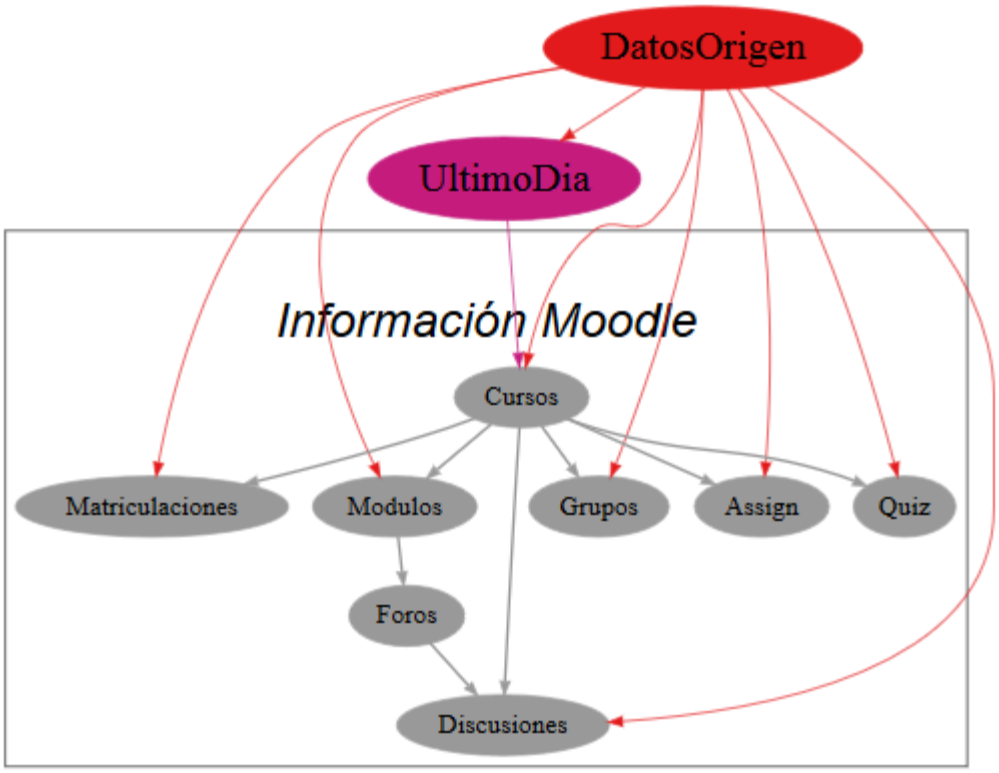


Figura 246. Detalle del conjunto de bloques en *Información Moodle* para la aplicación inicial

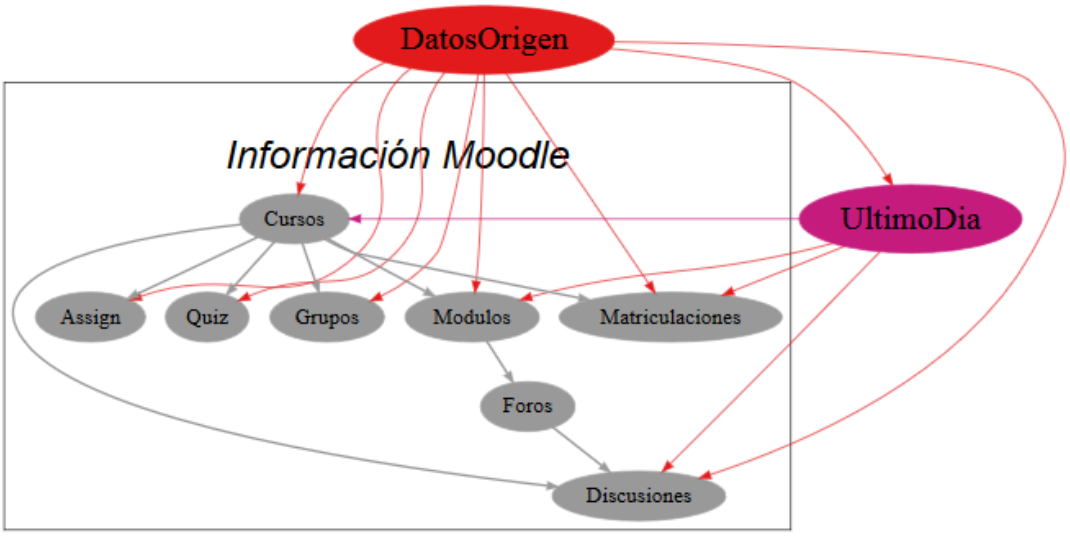


Figura 247. Detalle del conjunto de bloques en *Información Moodle* para la aplicación incremental

El conjunto ***Estadísticas de Acceso*** engloba los bloques encargados de la obtención de las estadísticas de acceso a *Moodle*, según lo planteado en la sección 4.6.3. Entre estos bloques podemos distinguir dos tipos:

- Bloques con resultados intermedios:
    - `CursoModuloUsuario`.
    - `CursoForoUsuario`.
    - `PlantillaDCUM`.
    - `PlantillaDCUF`.
    - `PlantillaDCUFD`.
    - `PlantillaDCU`.
    - `DiaSemana`.
    - `PrimeraVisita`.
    - `PorcAccesoModulos`.
    - `ModulosVisitadosC`.
    - `ModulosVisitadosCU`.
    - `ApuntesVisitadosC`.
    - `ApuntesVisitadosCU`.
    - `PorcDiscusionesAccedidasCUF`.
    - `PorcAccesoForos`.
    - `PorcForosAccedidosC`.
    - `PorcDiscusionesAccedidasC`.
    - `ForosVisitadosCU`.
    - `DiscusionesVisitadasCU`.
- Bloques con resultados finales sobre las estadísticas:
    - `EstDCUM`.
    - `EstDCUFD`.
    - `EstDCU`.
    - `EstDCUF`.
    - `EstCUM`.
    - `EstCM`.

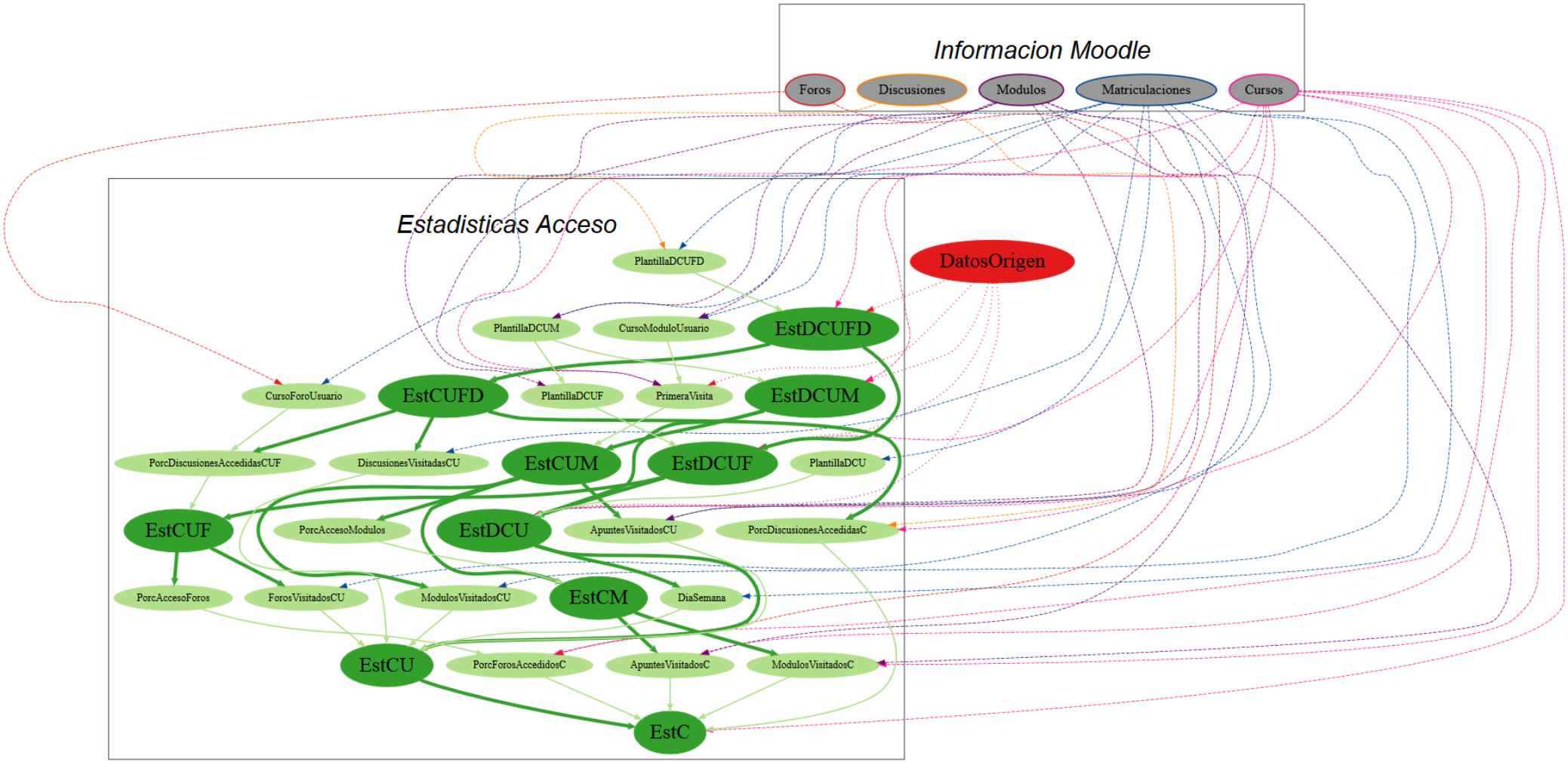


Figura 248. Detalle del conjunto de bloques en *Estadísticas de Acceso*

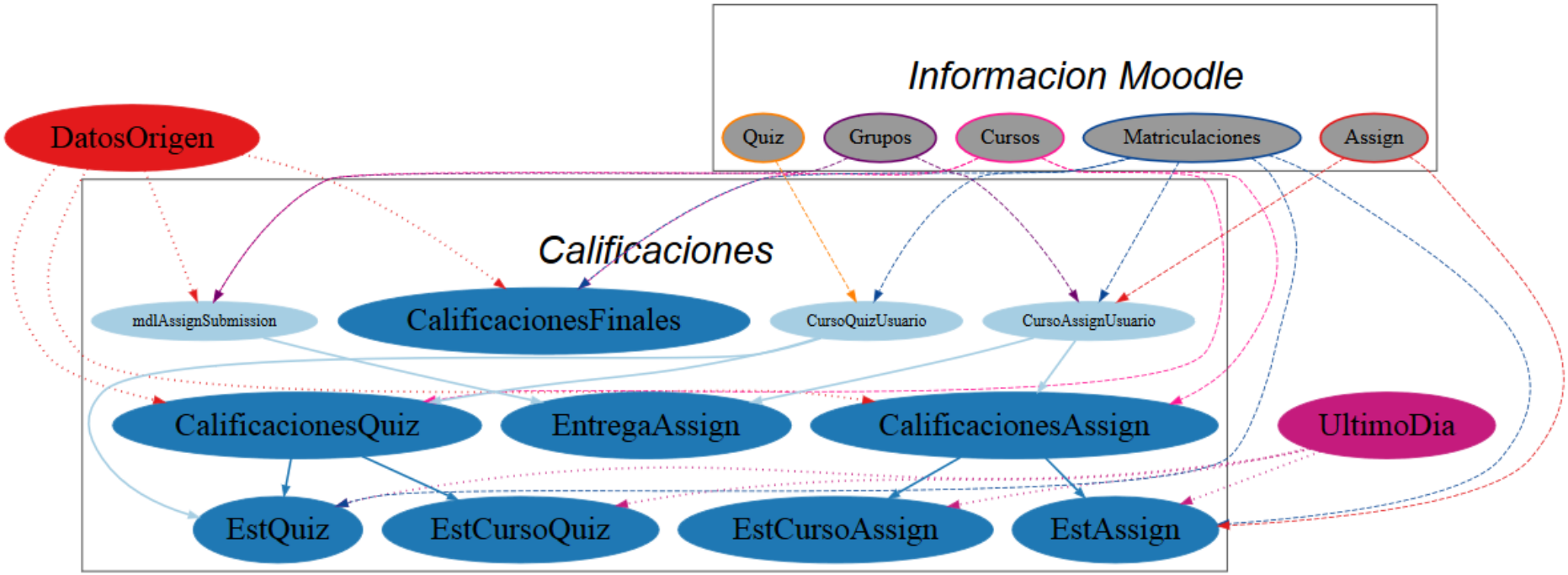


Figura 249. Detalle del conjunto de bloques en *Calificaciones* para la aplicación inicial

- `EstCUFD`.
- `EstCUF`.
- `EstCU`.
- `EstC`.

La relación entre estos bloques, así como con los de otros conjuntos se representa en la Figura 248, tanto para la carga inicial como incremental.

El conjunto de ***Calificaciones*** está formado por los bloques encargados de analizar los resultados académicos de los alumnos, tal y como se indicó en el apartado 4.6.4. En concreto, contará con dos tipos de bloques:

- Bloques intermedios para la obtención de resultados:
  - `CursoAssignUsuario`.
  - `CursoQuizUsuario`.
  - `mdlAssignSubmission`.
- Bloques finales con los resultados académicos:
  - `CalificacionesAssign`.
  - `EntregaAssign`.
  - `CalificacionesQuiz`.
  - `CalificacionesFinales`.
  - `EstAssign`.
  - `EstQuiz`.
  - `EstCursoAssign`.
  - `EstCursoQuiz`.

En la Figura 249 se detalla la obtención de este bloque para la aplicación inicial y en la Figura 250 para la aplicación incremental.

Bajo el conjunto ***Correlaciones*** se encuadran los bloques encargados de calcular la influencia entre la actividad en *Moodle* y los resultados académicos, según el apartado 4.6.5. Está compuesto por los siguientes bloques:

- `CorrCMA`.
- `CorrCMQ`.
- `CorrMNota`.
- `CorrMavgA`.
- `CorrMavgQ`.

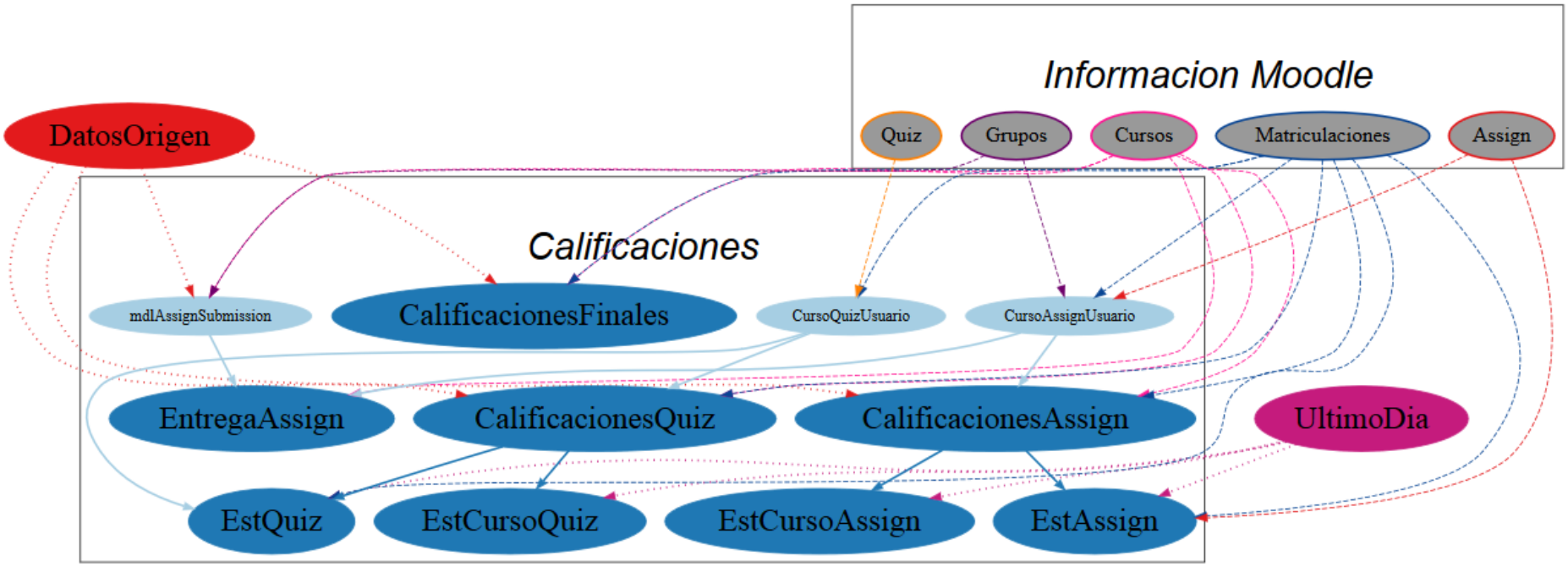


Figura 250. Detalle del conjunto de bloques en *Calificaciones* para la aplicación incremental

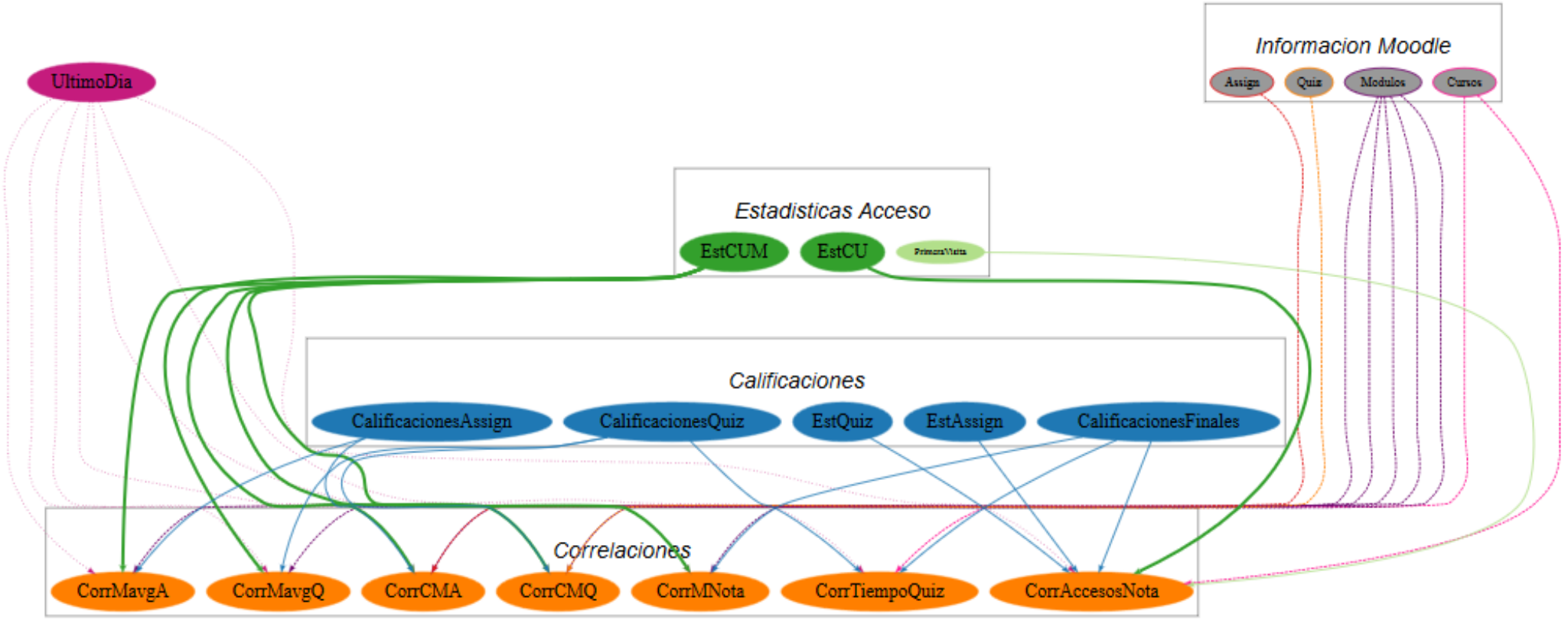


Figura 251. Detalle del conjunto de bloques en *Correlaciones*

- `CorrAccesosNota`.
- `CorrTiempoQuiz`.

La Figura 251 detalla el interior de este conjunto tanto para la aplicación inicial como la incremental.

Bajo el nombre ***Machine Learning*** no encontramos los bloques encargados de aplicar los algoritmos de *Clustering*, Reglas de Asociación y Clasificación enumerados en el apartado 4.6.6:

- `EstClustering`.
- `EstDiscretizadas`.
- `CursoClustering`.
- `CursoReglasAsociacion`.
- `CursoClasificacion`.

Su obtención se representa en la Figura 252 en la aplicación inicial y en la Figura 253 en la aplicación incremental.

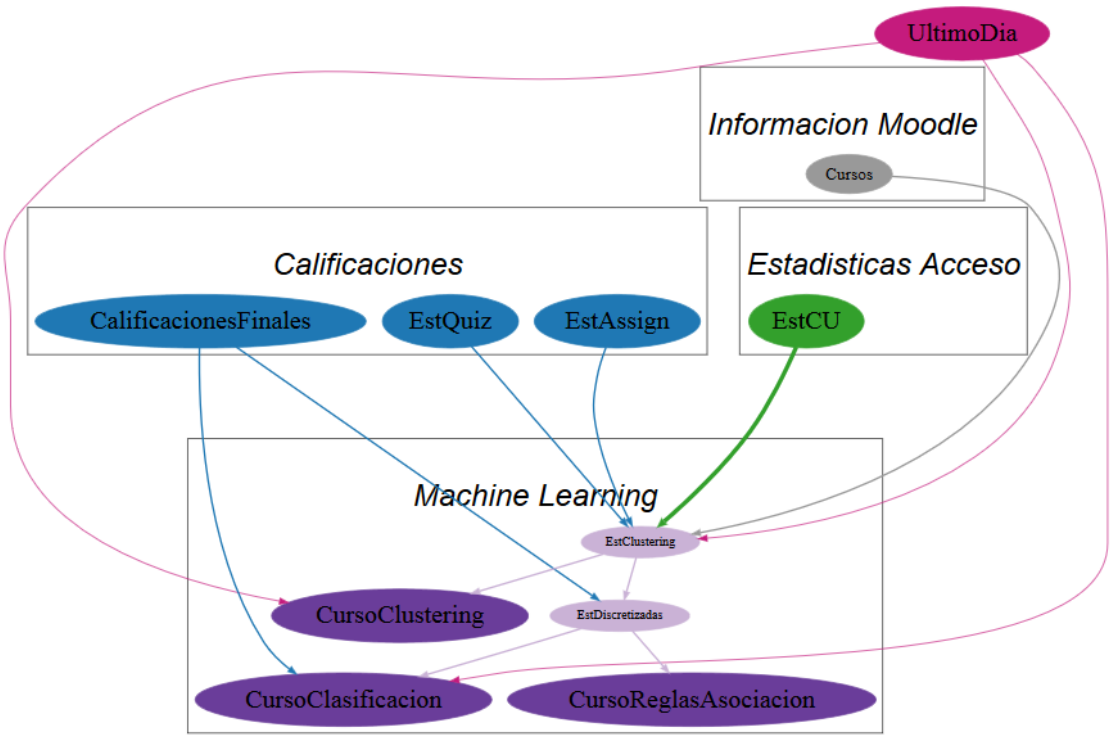


Figura 252. Detalle del conjunto de bloques en *Machine Learning* para la aplicación inicial

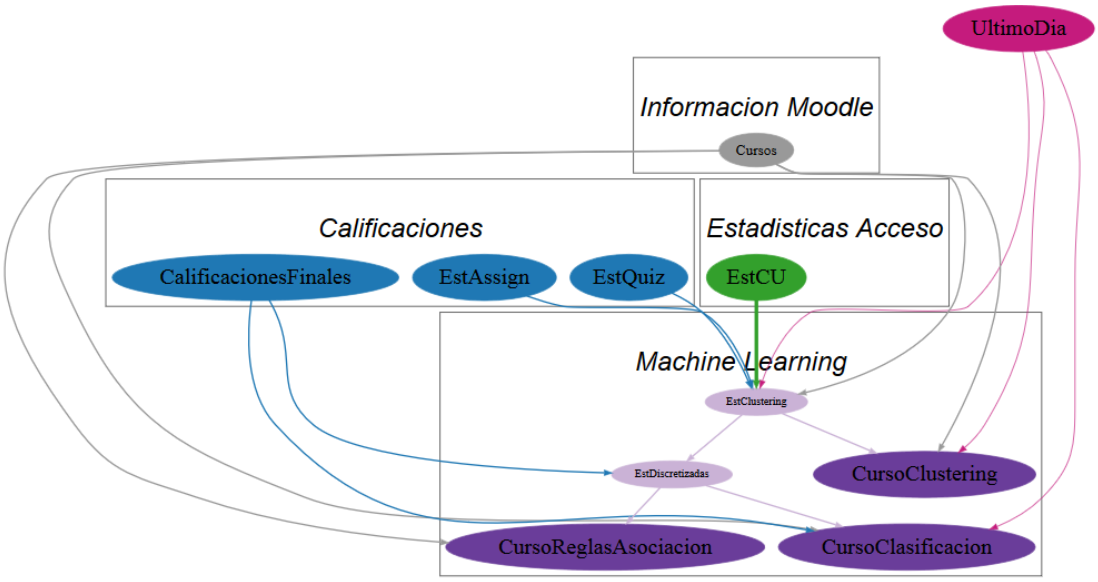


Figura 253. Detalle del conjunto de bloques en *Machine Learning* para la aplicación incremental

Por último, en el conjunto **Presentación** se engloban los bloques encargados de transferir la información correspondiente a *ElasticSearch* para su representación mediante *Kibana*, según el apartado 4.7.

- `CursosPres`.
- `ModulosPres`.
- `MatriculacionesPres`.
- `EstAssignPres`.
- `EstQuizPres`.
- `CalificacionesQuizPres`.
- `EstCMPres`.
- `EstCUPres`.
- `EstCPres`.
- `CalificacionesFinalesPres`.
- `CalificacionesAssignPres`.
- `EntregaAssignPres`.
- `EstDCUPres`.
- `EstCUMPres`.
- `CorrTiempoQuizPres`.
- `CorrAccesosNotaPres`.

- `CorrCMAPres`.
- `CorrCMQPres`.
- `CorrMNotaPres`.
- `CorrMavgAPres`.
- `CorrMavgQPres`.
- `CursoClasificacionPres`.
- `CursoClusteringPres`.
- `CursoReglasAsociacionPres`.

Su obtención se representa en la Figura 254.

4.9. Conclusiones y requisitos finales

En el Capítulo 3 se enumeró que el principal requisito del trabajo era adaptar los artículos de Casey, Gibson y Paris [10] y de Romero, Ventura y García [15] para replicar sus resultados mediante un infraestructura *BigData* modificando los aspectos que fueran necesarios. El resto de requisitos surgirían a medida que se realizase el trabajo y fueran surgiendo problemas.

Tras terminar el desarrollo de este capítulo se puede observar que se ha desarrollado una aplicación dividida en dos partes:

- Una aplicación inicial que permitirá inicializar el estudio.
- Una aplicación incremental que se ejecutará diariamente para actualizar el estudio.

El funcionamiento correcto de ambas está condicionado a los siguientes criterios:

- Los usuarios se identifican inequívocamente a través del campo `idnumber` de `mdl_user`. Este campo nos permitirá diferenciar un mismo usuario a lo largo de los años aunque cambie su identificador *Moodle*.
- Los cursos se identifican inequívocamente mediante el campo `idnumber` de `mdl_course`, el cual debe seguir la estructura indicada en el apartado 4.6.2. Este nos permitirá distinguir cursos aunque se reutilicen los identificadores *Moodle* así como relacionar un curso con otros cursos iguales en otros años académicos.
- Se trabaja bajo el supuesto que no se reutilizan identificadores de módulos o elementos contenidos en módulos, por lo menos, dentro del mismo curso.

La definición de estas aplicaciones ha dado lugar a una serie de requisitos que se podrían añadir a los indicados en el Capítulo 3:

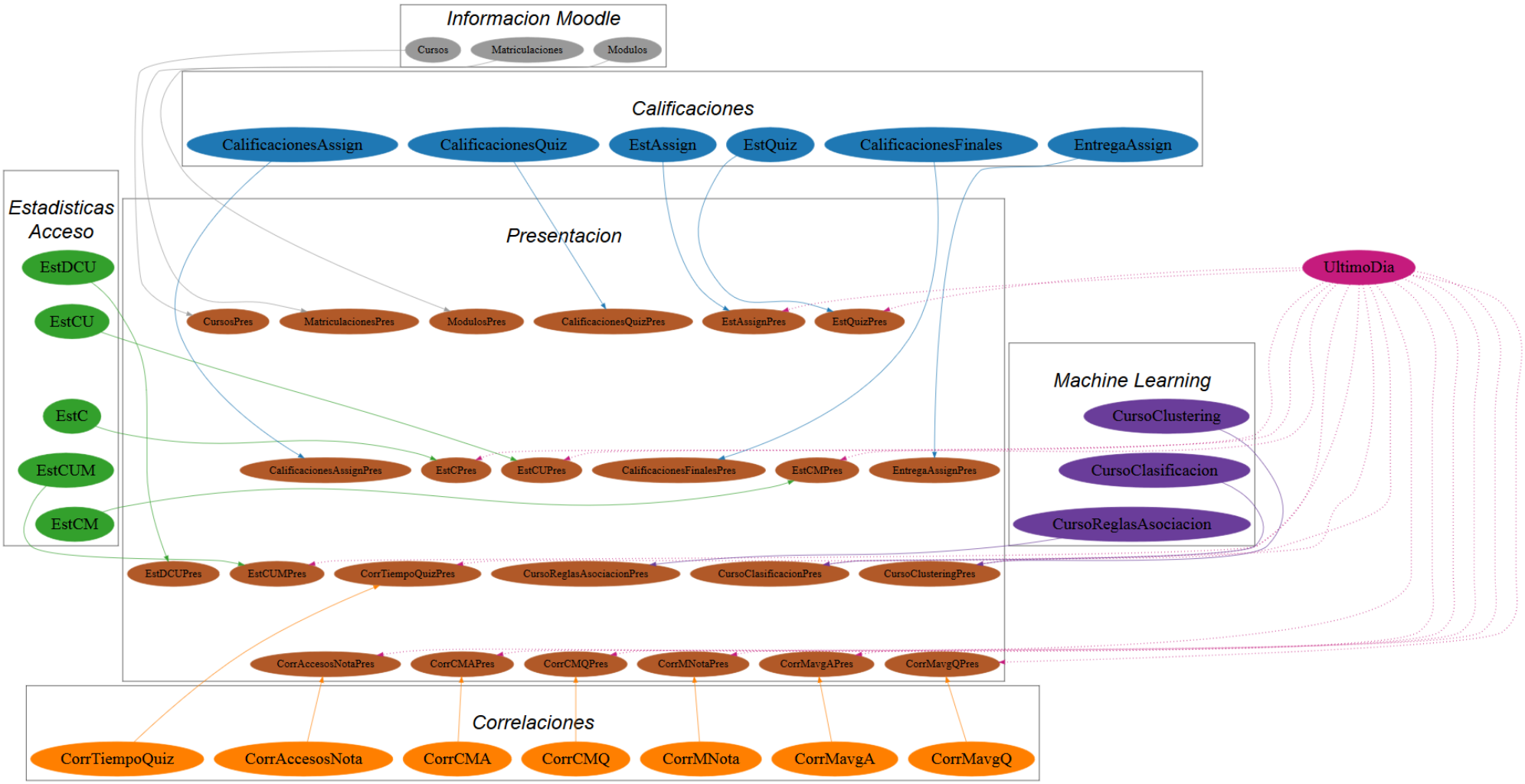


Figura 254. Detalle del conjunto de bloques en *Presentación*

- Se contará con dos aplicaciones diferenciadas, tal y como se acaba se enumerar.
- La aplicación incremental debe ser capaz de realizar su trabajo en menos de 24 horas, al menos de forma regular.
- Se debe configurar la ejecución de aplicaciones de tal forma que no se solapen en tiempo.

Por lo demás, a lo largo de este capítulo se ha podido comprobar la extensa lista de herramientas y formas de utilizar las herramientas que existen. Aquí solo se ha comentado aquella parte de estas herramientas que se han necesitado, pero lo manuales son aún más extensos.

Con respecto al trabajo realizado, se ha detallado con ejemplos el funcionamiento de los bloques de códigos dedicados a la primera parte del análisis directo sobre el contenido de *Moodle*, ya que parece ser la parte más complicada de entender y es importante comprender por qué hay que hacer cada paso o cómo se va generando el resultado final en cada etapa. Para no extender demasiado el texto, la obtención de métricas no se ha ilustrado con ejemplos, ya que su aplicación parece más intuitiva y entendible a partir del código fuente.

Por otro lado, tanto la estrategia planteada, las métricas y resultados obtenidos y la forma de obtener estos es totalmente susceptible de ser cuestionada. Probablemente se puedan encontrar otras formas de realizar más eficientes a nivel de ejecución o más fáciles a nivel de programación. El plano de configuración de la máquina también está abierto a debate, pero es una parte de cambio dinámico, por lo que se puede "jugar" con ella para encontrar aquellos parámetros que agilicen la ejecución.

A lo largo del desarrollo del este capítulo se han ido comentado puntos de posible mejora. Al empezar a trabajar con *BigData* se partió desde ningún conocimiento y se empezó poco a poco a programar. Con el paso del tiempo se fue obteniendo experiencia sobre la mejor forma de hacer las cosas, pero hay ciertos puntos en los que un cambio genera una enorme lista de cambios en cadena, por lo que se ha optado por dejarlo como está, por ejemplo, el punto en el que se cambia el identificador de curso *Moodle* por el *BigData* se podría haber hecho parejo al cambio de identificador de usuario, pero cuando se vio esta alternativa, era demasiado el trabajo realizado y el cambio necesario sería enorme.

En el siguiente capítulo, se realizará un análisis de las conclusiones generales del desarrollo de este trabajo.

Capítulo 5

Resultados y conclusiones

Para finalizar con el trabajo, se comentan las conclusiones obtenidas del mismo. Este capítulo se dividirá en los siguientes apartados:

- En el apartado 5.1 se concluirá el cumplimiento de los requisitos planteados en el Capítulo 3.
- En el apartado 5.2 se resumirá en qué consiste la aplicación diseñada.
- En el apartado 5.3 se desarrollará cómo ha sido la ejecución práctica de la aplicación, problemas encontrados y conclusiones extraídas.

5.1. Cumplimiento de los requisitos

En el Capítulo 3 se planteó como requisitos adaptar los artículos [10], [15] a *BigData*. El trabajo ha conseguido realizar esta adaptación, pero con alguna modificación:

- Siguiendo el primer artículo se pretendía calcular la influencia entre diversos elementos de *Moodle* por medio de la correlación de *Pearson*. Se ha aplicado en prácticamente todos los elementos enumerados en el artículo y se ha añadido el cálculo de influencia entre otras tantas métricas, para añadir más valor al resultado, como se explicó en el apartado 4.6.5.
- En cuanto al segundo artículo, se han aplicado algoritmos de *clustering*, reglas de asociación y clasificación con los mismos objetivos, modificando ligeramente las métricas utilizadas para ello, según se comentó en el apartado 4.6.6.

Por lo tanto se puede confirmar la posibilidad de aplicar *BigData* para la obtención de resultados similares a otras técnicas.

5.2. Resumen de la aplicación

La aplicación resultado de este trabajo realmente se compone de **dos aplicaciones**. La primera es una aplicación inicial a ejecutar en el momento de poner en marcha el análisis *BigData* sobre *Moodle*. Se encargará de inicializar todo el contenido correspondiente, así como realizar

el primer análisis. La otra aplicación será incremental y bastante similar a la anterior, se encargará de realizar un análisis diario de las modificaciones o nuevos datos que encontramos en *Moodle* para añadirlos al análisis anterior. Es requisito que esta aplicación sea capaz de analizar *Moodle* en menos de 24h ya que, en caso contrario, no se podrá realizar un análisis diario.

Cada una de estas aplicaciones están compuestas por una serie de bloques de ejecución, indicados en el Capítulo 4, cuyo lanzamiento está controlado por *Oozie*. Esta herramienta se encargará tanto de controlar la ejecución ordenada de los bloques (según las dependencias entre ellos) como el calendario de lanzamiento diario de la aplicación, además de gestionar posibles ejecuciones solapadas.

Estos bloques de ejecución tendrán por objetivo implementar **tres etapas de procesado**:

- Carga de datos y preprocesado previo de la información de *Moodle*, desde la base de datos que corresponda, usando *Sqoop* y *Hive*.
- Análisis *BigData*, según los artículos referencia, usando, principalmente, *Hive* para realizar los cálculos correspondientes.
- Transferencia de los resultados a *ElasticSearch* para su representación gráfica mediante *Kibana*.

Mediante la **representación gráfica** de los resultados obtenidos (apartado 4.7) se pueden comprobar rápidamente cuestiones interesantes, más allá del nivel de resultados estadísticos, como:

- Existencia de módulos, actividades, elementos, etc. que contribuyen negativamente en el aprendizaje de los alumnos por tener una correlación negativa.
- Posibilidad de estimar la nota que obtendrán los alumnos, prestando especial atención a los casos negativos (abandonos o suspensos), para intentar corregirlos.
- Existencia de alumnos con comportamiento marginal, por ejemplo, por medio del sistema de predicción de nota o por medio del mecanismo de *clustering*.
- Patrones de comportamiento de los alumnos, por ejemplo, si un alumno tiene un nivel bajo de apuntes leídos, el nivel de *assign* aprobados es alto. Así también se pueden encontrar comportamientos anómalos.
- Indicar al profesor que tiene alumnos sin calificar, para que este busque los motivos.

5.3. Ejecución práctica

Con respecto a la ejecución práctica de las aplicaciones, podemos indicar muchos puntos. Para la prueba de las aplicaciones implementadas se ha usado una versión propia de *Moodle* del laboratorio *EdUVaLab* la cual apenas contaba con **datos**, entre otros:

- 9 cursos.
- Menos de 180 relaciones curso – usuario.
- Menos de 400 módulos definidos entre todos los cursos.

En definitiva una cantidad de datos que no recomendaría usar *BigData* porque con la propia base de datos *SQL* de *Moodle* y herramientas de *Data Mining* se podría realizar el trabajo.

Pero esta ha sido la base de datos de prueba, principalmente para comprobar manualmente la validez de los resultados, que las métricas se calculan correctamente o que la ejecución condicional por dependencias de *Oozie* sigue el orden adecuado. Realizar estas comprobaciones con una base de datos real podría ser muy laborioso y requerir de mucho tiempo.

A pesar de contar con tan pocos datos, los **tiempos de ejecución** han sido muy decepcionantes ya que la ejecución de las sentencias *HiveQL* definidas requiere de más tiempo que si esa sentencia se ejecutara en una base de datos *SQL* normal. Por ejemplo, simplemente, y como caso muy significativo, la ejecución de una sentencia `SELECT` para visualizar el contenido de una tabla, en la base de datos *MySQL* de *Moodle* se realiza de forma inmedianta, mientras que en *Hadoop* puede llegar a tardar entre 10 y 30 segundos. Obviamente, este empeoramiento de rendimiento puede ser debido a que la mayor parte del tiempo se consuma en la inicialización de una infraestructura que se está preparando para recibir muchos datos, aunque vaya a recibir pocos. En varios foros de usuarios dedicados a la programación en *BigData* comentan que no tiene sentido usar *Hive* para pocas cantidades de datos ya que el rendimiento va a ser muy malo comparado con cualquier otra base de datos *SQL*, por lo que podemos explicar los tiempo de ejecución obtenidos.

Otra cuestión planteada es, ¿qué pasará cuando utilicemos datos de una base de datos real?, es decir, cuando tengamos realmente **grandes cantidades de datos**. La lógica de cualquier ejecución hace pensar que el tiempo de ejecución se incrementará linealmente con la cantidad de datos. En la práctica esto parece que no ocurre. Se han realizado un par de pruebas muy leves simulando tener una cantidad de datos similar a la que tendríamos en el *Moodle* de la UVa y parece que sí que existe un incremento, pero muy inferior a lineal. Podemos tratar de explicarlo con algún ejemplo. Tener en cuenta que los tiempos de ejecución pueden variar mucho dependiendo de la situación, lo que interesa son los incrementos relativos.

- Pasar de una tabla de matriculaciones de 166 relaciones a más de 700.000 ha incrementado el tiempo de ejecución en menos de 20 segundos (sobre el minuto original), aunque sobre todo se incrementa el tiempo al tener que guardar los datos.

- Las operaciones más costosas de implementar son las sentencias `LATERAL VIEW` y los cruces *JOIN*; sobre todo la primera, se recomienda evitarla siempre que se pueda. En este trabajo se ha utilizado `LATERAL VIEW` para generar las plantillas temporales (apartado 4.6.1). Ya se indicó que esta generación es uno de los puntos más conflictivos del análisis. En el mismo caso anterior, pasar de una tabla de matriculaciones de 166 registros a un valor entre 700.000 y 800.000 registros hace pasar la plantilla DCU de 63000 registros a más de 265 millones. El primer caso se ejecuta en un tiempo de alrededor de un minuto, mientras que el segundo se ejecutó en un tiempo entre hora y media y dos horas (promedio de las distintas pruebas). Por lo tanto, la cantidad de datos se multiplicó por un factor aproximado de 4200, pero el tiempo se multiplicó por un factor inferior, 120.

- Hacer una operación de `GROUP BY` con suma de una métrica de los 265 millones de registros simulados anteriores llevó un tiempo de entre 10 y 15 minutos, mientras que con los 63000 reales, lleva alrededor de 1 minuto.

Si la ejecución con nuestros pocos datos ya parecía lenta, las cosas se empeoraron mucho más al ejecutarlo a través de ***Oozie*** tras haberlo automatizado. Recordando la explicación de la ejecución de los *workflows* del apartado 4.8.3, por ejemplo, para lanzar una acción *Hive* antes

debe lanzar un *Launcher* que prepara la ejecución de la acción que le sigue. Obviamente esto consume un tiempo, de tal forma que si la ejecución de las sentencias *Hive* es muy rápida, este tiempo del *Launcher* puede llegar a ser el tiempo predominante en la ejecución. Además, se debe tener en cuenta que si un bloque de ejecución tiene varias acciones en paralelo o secuenciales cada una lleva su *Launcher*, por lo que aumenta el tiempo no efectivo de ejecución.

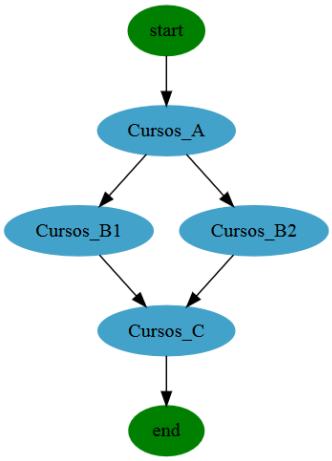


Figura 255. Detalle del bloque de ejecución `Cursos`

Por ejemplo, supongamos que tenemos el bloque `Cursos`. Su ejecución interna tiene la estructura que se muestra en la Figura 255. Está compuesto por cuatro acciones *Hive*, remarcadas en azul. En la Tabla 145 se muestran los tiempos de ejecución obtenidos en una simulación del bloque con un total de 12 cursos.

Script	Tiempo de ejecución (segundos)
`Cursos_A`	42,708
`Cursos_B1`	60,785
`Cursos_B2`	104,221
`Cursos_C`	16,511
Total	224,225

Tabla 145. Ejemplos de tiempo de ejecución del bloque `Cursos`

El bloque ha tardado en ejecutarse un total de 5m19s = 319s, es decir, aparecen 94,775s de tiempo de ejecución en el que no se ha realizado ningún trabajo, por lo que la eficiencia de ejecución es del 70%. Si tenemos en cuenta que los *scripts* `Cursos_B1` y `Cursos_B2` se ejecutan en paralelo, realmente el tiempo de ejecución efectiva se vería disminuido, ya que de entre estos dos tiempos, escogeríamos el mayor, por ser el más restrictivo, para un total de 163,44s de ejecución efectiva y un rendimiento de ejecución del 51%. En definitiva se pueden observar los dos problemas planteados:

- El tiempo de ejecución es bastante lento, teniendo en cuenta la escasez de datos y comparándolo con otras herramientas *SQL*.
- El lanzamiento a través de *Oozie* aumenta los tiempos de ejecución por la ejecución del *Launcher* previo al inicio de la ejecución efectiva de las sentencias *HiveQL*, las cuales tardar en ejecutarse lo mismo que si se lanzan manualmente contra el servidor Hive.

Con respecto al primer problema, podemos analizar un poco más en detalle qué ocurre y qué tiene que hacer la sentencia para ver si encontramos sentido a los tiempos de ejecución.

El *script* `Cursos_A` realiza dos cálculos, guardando los correspondientes resultados en sendas tablas:

- Accede a `mdl_course` y para cada entrada toma los campos `id` e `idnumber`.
- Genera el identificador *BigData* de los cursos *META* como concatenación de los identificadores *BigData* de los cursos hijos. Para ello, accede a `mdl_enrol` y toma los cursos que tienen matriculaciones de tipo *META* para identificarlos como cursos agrupación. Cruza este resultado con la tabla de cursos `mdl_course`, para obtener el `idnumber` que le corresponde a cada curso hijo del curso META. Cuando se tiene esta información con el identificador *BigData* para el curso *META*, se cruza con el resultado anterior para a cada curso *META* asociarle el identificador *BigData* que vamos a usar.

El primer cálculo es muy sencillo, y el segundo es un poco más complicado, pero dada la cantidad de datos (12 cursos, para simular cursos hijos) podría esperarse un tiempo menor.

El *script* `CURSOS_B1` construye la tabla de cursos actuales en dos pasos:

- De la primera tabla de `CURSOS_A` escoge aquellas entradas que no están en el segundo resultado del mismo *script* (cursos normales).
- De la primera tabla de `CURSOS_A` selecciona aquellas entradas que sí están en la segunda tabla (cursos *META*), a la vez que modifica su `idnumber` por el identificador *BigData* que se ha construido en el segundo resultado, implementado mediante un cruce de tablas con `CROSS JOIN`.

La primera operación no debería conllevar mucho tiempo, ya que es copiar menos de 12 registros y para cada uno, dos campos. La segunda, necesitaría un poco más de tiempo por tener que cruzar tablas y guardar un *bucket*.

El *script* `CURSOS_B2` construye la tabla de histórico de cursos en tres pasos:

- De `mdl_course` se toma la información de curso que corresponde para aquellos cursos que no son *META*, en base al segundo resultado del *script* `CURSOS_A`.
- De `mdl_course` se toma la información de curso que corresponde para aquellos cursos que son *META*, en base al segundo resultado del *script* `CURSOS_A`, a la vez que se modifica su `idnumber` por el identificador *BigData* construido en este resultado, implementado mediante un cruce de tablas con `CROSS JOIN`.
- Los dos resultados previos se guardan en una misma tabla, a partir de la cual se construye la tabla final. En ella se insertan directamente los cursos de la tabla anterior, siempre que no estuvieran ya añadidos, y a la vez se modifica su fecha de inicio, por si no es coherente, en base a una serie de criterios especificados en el código fuente. Para ello:
    - De la tabla temporal con los cursos de los dos primeros pasos se seleccionan solo las entradas cuyo identificador de curso (que ya es *BigData*) no esté añadido ya a la tabla de histórico de cursos.
    - Este resultado se cruza con la tabla de `ultimo_dia`. En caso de que la fecha de creación del curso sea anterior a un determinado valor almacenado en esta tabla, la fecha de creación del curso se sustituye por esta fecha.

De forma análoga al *script* anterior, las dos primeras sentencias deberían ejecutarse directamente, y la tercera tardar un poco más por tener que cruzar tablas y guardar el resultado haciendo un *bucket* de los valores. Comparado con `Cursos_B1`, se esperaría que tardase más, como ocurre, porque tiene mayor complejidad de programación.

El *script* `Cursos_C` simplemente actualiza el valor de la fecha previamente comentada mediante un `DELETE` del registro actual y un `INSERT` con la nueva fecha, la cual se escoge de otro registro de esa misma tabla. Algo que en principio debería hacerse en un tiempo mucho menor a los 16s que tarda en la práctica.

Cualquiera de estos tiempos no se ven notablemente incrementados si se incrementa la cantidad de cursos, de forma similar a como ocurría con la tabla de matriculaciones.

Si nos vamos a un contexto más general, analizamos la **ejecución total** de todos los bloques según el orden secuencial de dependencias que le corresponden. En este punto, los tiempos totales de ejecución son muy relativos ya que dependen de varios aspectos:

- Situación actual de la máquina real donde se esté ejecutando la máquina virtual.
- Memoria *RAM* asignada a *YARN*, en la máquina virtual. Cuanta más memoria se tenga asignada, más acciones podrá realizar en paralelo.
- Capacidad de la máquina donde se esté ejecutando la máquina virtual.

Se ha probado a ejecutar las aplicaciones en dos ordenadores distintos y hay diferencias muy apreciables. Uno de ellos, es el ordenador del laboratorio y otro, un ordenador personal. Las características se muestran en la Tabla 146. En el ordenador del laboratorio, no se han conseguido tiempos de ejecución inferiores a 6 horas. Mientras que en el ordenador personal, el mejor de los registros está alrededor de las 4 horas (teniendo en cuenta que realiza las cosas de forma secuencial al no tener apenas *RAM*). En cualquiera de los casos, tiempos de ejecución muy altos teniendo en cuenta la poca cantidad de datos con la que se realiza las pruebas.

	Ordenador de laboratorio	Ordenador personal
Sistema operativo	*Windows* 10	*Windows* 7
Programa de virtualización	*Virtual Box*	*VMWare*
CPU	*AMD A10-5800K* • 4 núcleos • Virtualización de *AMD*	*Intel i5-4590* • 4 núcleos • Virtualización de *Intel*
RAM	32 GB	12 GB
***RAM* asignada a la MV**	25 GB	9 GB
RAM* asignada a *YARN	20 GB	7 GB

Tabla 146. Características de los ordenadores utilizados para las pruebas de ejecución

Se ha analizado un poco más en concreto los tiempos de ejecución en el ordenador personal, por proporcionar, aparentemente, mejores resultados, no solo a nivel de ejecución global, sino también a la hora de ejecutar sentencias simples desde la consola de *Hive*, (se ejecutan más rápido). En concreto, se ha programado una ejecución completa de la aplicación inicial y otra de la incremental, obteniéndose los resultados de la Tabla 147. El tiempo total de ejecución de los bloques se ha calculado sumando el tiempo que ha tardado cada sentencia *HiveQL* programada. Este tiempo se puede tomar como referencia del tiempo que tardaría en ejecutarse la aplicación si se realizase todo de forma secuencial. Cabe la duda de si al paralelizar sentencias, estas se puedan ver ligeramente ralentizadas al tener que compartir recursos con

otras sentencias, pero se ha comparado, aleatoriamente, el tiempo de ejecución de una sentencia cuando se ejecuta en paralelo con otra y cuando se ejecuta sola, y no se aprecia un incremento notable, simplemente una ligera variación, que puede estar debida a la propia variación del estado de la máquina entre las dos ejecuciones. Sí se ha notado un incremento más notable cuando se ejecutan varios programas *Java* en paralelo, en este caso, se ha tomado el tiempo que se ha visto tardan cuando se ejecutan solos desde consola, como tiempo a sumar a la ejecución secuencial.

	inicial	incremental
tiempo de ejecución secuencial	6h18m = 22696,3732s	5h48m = 20896,9745s
tiempo de ejecución paralela mediante *Oozie*	5h26m = 19560s	4h26m = 15960s
eficiencia en la ejecución	116%	130%

Tabla 147. Eficiencia de la ejecución completa de las aplicaciones

Se pueden observar dos cosas:

- Los tiempos de ejecución siguen siendo altos para los pocos datos que tenemos.
- A pesar de que estos tiempos de ejecución se ven aún más incrementados tras lanzar las sentencias *HiveQL* (y demás herramientas) a través de *Oozie*, por la presencia de los *Launcher*, al ejecutar todo el conjunto de bloques, se puede ver que se obtiene muy buena eficiencia, debido a que mientras una acción está en la etapa de lanzar el *Launcher*, otra estará ejecutando su trabajo, por lo que esta generación de *Launcher* parece camuflarse.

Así, en principio, la eficiencia deja ser un problema, pero el tiempo que se tarda en ejecutar las aplicaciones, sigue siendo conflictivo.

De cualquiera de los casos se puede extraer, que parece que una simple ejecución como la programada (simple a nivel de datos, no a nivel de flujo de actividades) tiene unos requisitos muy grandes, más de lo pensado, dada las garantías teóricas del *BigData*.

Obviamente, tras todo lo planteado en este capítulo se pone seriamente en duda la puesta en marcha real de estas aplicaciones, ya que parece conducir a unos tiempos de ejecución que impedirían la ejecución diaria de la aplicación incremental. Se puede optar por varias **alternativas**:

- Tratar de bucear por los cientos de parámetros de configuración de *Hadoop* para buscar posibles errores o valores que entorpezcan la ejecución.
- Generar un *cluster* con más máquinas, ya que parece que una sola máquina no tiene capacidad suficiente. Esta opción suma más problemas, que de forma teórica implicarían un aumento de los tiempos: gestionar varias máquinas y trasladar la información entre ellas.
- Mantener una sola máquina pero en la que se reúna una *CPU* potente y una buena cantidad de memoria *RAM*.
- Modificar el modo de ejecución de las tareas, por ejemplo, tratar de secuenciarlo más o no dividir el trabajo en tantos bloques ejecutables. Dados los valores de eficiencia, parece que la paralelización no es un problema, pero se puede explorar como vía de posible mejora.

Capítulo 6

Líneas futuras

Para finalizar la exposición del trabajo se presentan posibles acciones futuras que se pueden realizar relacionadas con él.

En primer lugar, relativo a la **codificación** de las sentencias *HiveQL* y programas *Java* para la obtención de resultados, podría citarse:

- Las sentencias *HiveQL* son susceptibles de mejoras a nivel de optimización, por ejemplo, en el orden en que se realizan las cosas o definiendo de otra forma el *bucket* y particionado de las tablas.

- Modificar el punto del código en que el identificador *Moodle* del curso se sustituye por el identificador *BigData*, como se comentó en el apartado 4.6.2.

- Modificar los ejecutables *Java* para la aplicación de *Machine Learning* de forma que sean parametrizables, necesitando, además, añadir una etapa de validación de los parámetros de entrada, como se explicó en el apartado 4.6.6.

En segundo lugar, dadas las conclusiones del Capítulo 5, se prevé que la aplicación tal y oómo está no se podrá aplicar a datos reales sobre la infraestructura diseñada, por el tiempo que supondrá la ejecución, por lo que es muy posible que se tenga que **replantear la estrategia**, por ejemplo:

- Eliminar métricas que no tienen mucha utilidad.

- Reducir el número de métricas necesarias o utilizadas para la obtención de resultados de interés (correlación y *Machine Learning*).

- No realizar un análisis tan detallado, por ejemplo, la creación de las plantillas en el análisis inicial puede llevar mucho tiempo. Se ha estimado que en el cálculo incremental no serán conflictivas. Se pueden eliminar estas plantillas y realizar un análisis más leve, recordando el por qué se utilizan estas plantillas (apartado 4.6.1).

- Dado el comportamiento, sobre todo a nivel de tiempo necesitado, de la automatización de *Oozie* observada en la práctica (apartado 5.3), como ya se comentó podría ser necesario reducir el número de bloques definidos para que engloben acciones más generales y no tan particulares, o secuenciar más las dependencias de ejecución.

- La importación de datos desde la base de datos origen de *Moodle* por medio de *Sqoop* es la etapa más crucial a nivel de tiempo de ejecución, se puede tratar de buscar otras formas de implementarla así como modificar el planteamiento aquí utilizado, el cual sobrescribe toda la información de las tablas anteriormente importadas.

También se debe explorar la **configuración global de la máquina** y la opción de generar un *cluster* con más de una máquina, así como tantear la posibilidad de implementar una máquina *Hadoop* real, en vez de hacerlo por medio de una máquina virtual, para tratar de comparar el rendimiento y extraer conclusiones sobre la influencia de la virtualización de *Hadoop* en el rendimiento. En este sentido, existen numerosos estudios [219]–[221] comparando el rendimiento entre un *cluster* físico y otro virtual. En muchos casos, se recomienda no virtualizar *Hadoop*, ya que las opciones de virtualización que ofrecen las diversas compañías están orientadas al aprendizaje de la infraestructura, pero no a poner en marcha una infraestructura de cálculo intensivo. Existen diversas comparativas en las que se indica que si el cálculo implementado lleva implícito muchas operaciones de escritura (como es nuestro caso) el rendimiento se ve fuertemente empeorado [222]. También afecta el sistema de virtualización que utilizamos. *VMWare* ofrece *vSphere* como mejor opción para virtualizar *Hadoop* [223], siendo una de las plataformas que más se recomiendan para este objetivo.

Relativo a la **explotación de la infraestructura**, no se ha incidido en este trabajo, por estar utilizando una máquina para probar los algoritmos diseñados, pero debería tratarse la gestión de errores y recuperación en caso de fallo, tanto de ejecución de las aplicaciones como recuperación de desastres en la máquina. Por otro lado, *Hadoop* proporciona un servicio denominado *Ranger* que se encargará de la gestión de seguridad del sistema. En este trabajo, ese servicio se ha desactivado por simplicidad en el desarrollo de operaciones. Las nuevas versiones de la *sandbox* de *Hortonworks* son más restrictivas con este servicio y obligan a cambiar contraseñas por defecto, autenticación de usuarios para realizar tareas que en versiones anteriores podía realizar cualquiera, etc.

Además, la **capa de presentación** de resultados debe ser replanteada. Aquí se ha utilizado la combinación *ElasticSearch* y *Kibana* como etapa de verificación de los resultados obtenidos y para poder demostrar gráficamente los resultados de este trabajo. Esta etapa no se puede implementar en la práctica, ya que lo ideal sería plantear una interfaz en la que cada profesor pudiera ver la actividad de cada uno de sus cursos. *Kibana* no proporciona ninguna posibilidad de implementar esto:

- No proporciona mecanismos de autenticación para diferenciar usuarios.
- No hay forma de aislar la información de un curso a profesores de otros cursos.

Si utilizásemos *Kibana*, definiríamos un panel con la información gráfica correspondiente, en el cual habría que filtrar los datos por medio del identificador del curso que le corresponda al profesor, pero dado el funcionamiento de los paneles de *Kibana*, este filtro podría ser modificado por el profesor. Además, el uso de estos filtros puede ser complicado de entender por un profesor.

Por todo ello, será conveniente generar una interfaz que se ajuste a los requisitos de la información que realmente se quiere mostrar. En la etapa de presentación desarrollada en el trabajo la información generada en *BigData* ha tenido que ser, en algunos casos, transformada, ya que en su forma original las posibilidades de representación mediante *Kibana* eran mínimas. Así, desarrollando una aplicación de presentación propia, esta se ajustará a la forma y contenido de los resultados *BigData*. En el apartado 4.7.2 se proporcionaron ejemplos de representaciones gráficas que se pueden generar en esta interfaz.

Finalmente, el análisis diseñado se puede ampliar para añadir la obtención de **nuevos resultados** como los planteados a lo largo del apartado 2.1.2. Obviamente, este trabajo deberá realizarse una vez se haya solventado la problemática del tiempo de ejecución, y se garantice una ejecución diaria inferior a las 24 horas.

Referencias

[1] A. Gandomi y M. Haider, «Beyond the hype: Big data concepts, methods, and analytics», *Int. J. Inf. Manag.*, vol. 35, n.º 2, pp. 137–144, 2015.

[2] D. Beaver, S. Kumar, H. C. Li, J. Sobel, y P. Vajgel, «Finding a Needle in Haystack: Facebook's Photo Storage.», en *OSDI*, 2010, vol. 10, pp. 1–8.

[3] Facebook, «Scaling Facebook to 500 Million Users and Beyond». [En línea]. Disponible en: https://www.facebook.com/notes/facebook-engineering/scaling-facebook-to-500-million-users-and-beyond/409881258919. [Accedido: 20-oct-2015].

[4] iTnews, «Inside eBay's 90PB data warehouse - Software - Storage». [En línea]. Disponible en: http://www.itnews.com.au/news/inside-ebay8217s-90pb-data-warehouse-342615. [Accedido: 14-oct-2015].

[5] HowStuffWorks, «Amazon Technology». [En línea]. Disponible en: http://money.howstuffworks.com/amazon1.htm. [Accedido: 14-oct-2015].

[6] SearchEngineland.com, «Google Still Doing At Least 1 Trillion Searches Per Year». [En línea]. Disponible en: http://searchengineland.com/google-1-trillion-searches-per-year-212940. [Accedido: 14-oct-2015].

[7] Climate Change Simulation, «NASA's Weather Supercomputer». [En línea]. Disponible en: http://www.csc.com/cscworld/publications/81769/81773-supercomputing_the_climate_nasa_s_big_data_mission. [Accedido: 14-oct-2015].

[8] Moodle, «Moodle - Open-source learning platform». [En línea]. Disponible en: https://moodle.org/. [Accedido: 14-oct-2015].

[9] Moodle, «Moodle.org: Moodle Statistics». [En línea]. Disponible en: https://moodle.net/stats/. [Accedido: 14-oct-2015].

[10] K. Casey, P. Gibson, y I.-S. Paris, «Mining Moodle to understand Student Behaviour», en *International Conference on Engaging Pedagogy 2010 (ICEP10), National University of Ireland Maynooth*, 2010.

[11] J. McCarthy y E. A. Feigenbaum, «In Memoriam: Arthur Samuel: Pioneer in Machine Learning», *AI Mag.*, vol. 11, n.º 3, p. 10, 1990.

[12] C. Romero, S. Ventura, y C. Hervás, «Estado actual de la aplicación de la minería de datos a los sistemas de enseñanza basada en web», *Actas III Taller Nac. Min. Datos Aprendiz. TAMIDA2005*, pp. 49–56, 2005.

[13] B. M. Galán y D. R. Mateos, «La evaluación de la formación universitaria semipresencial y en línea en el contexto del EEES mediante el uso de los informes de actividad de la plataforma Moodle», *RIED Rev. Iberoam. Educ. Distancia*, vol. 15, n.º 1, 2012.

[14] U. Fayyad, G. Piatetsky-Shapiro, y P. Smyth, «From data mining to knowledge discovery in databases», *AI Mag.*, vol. 17, n.º 3, p. 37, 1996.

[15] C. Romero, S. Ventura, y E. García, «Data mining in course management systems: Moodle case study and tutorial», *Comput. Educ.*, vol. 51, n.º 1, pp. 368–384, 2008.

[16] Tutorials Point, «Mahout Tutorial». [En línea]. Disponible en: http://www.tutorialspoint.com/mahout/index.htm. [Accedido: 22-oct-2015].

[17] R. Agrawal, T. Imieliński, y A. Swami, «Mining association rules between sets of items in large databases», en *ACM SIGMOD Record*, 1993, vol. 22, pp. 207–216.

[18] F. Dang Ngoc y F. Dang Ngic, «Finding association rules with Mahout Frequent Pattern Mining», *Chimpler*. .

[19] saedsayad, «Association Rules». [En línea]. Disponible en: http://www.saedsayad.com/association_rules.htm. [Accedido: 31-oct-2015].

[20] R. Agrawal, R. Srikant, y others, «Fast algorithms for mining association rules», en *Proc. 20th int. conf. very large data bases, VLDB*, 1994, vol. 1215, pp. 487–499.

[21] J. Han, J. Pei, y Y. Yin, «Mining frequent patterns without candidate generation», en *ACM SIGMOD Record*, 2000, vol. 29, pp. 1–12.

[22] X. Chen, M. Vorvoreanu, y K. P. Madhavan, «Mining social media data for understanding students' learning experiences», *Learn. Technol. IEEE Trans. On*, vol. 7, n.º 3, pp. 246–259, 2014.

[23] C. Romero, P. G. Espejo, A. Zafra, J. R. Romero, y S. Ventura, «Web usage mining for predicting final marks of students that use Moodle courses», *Comput. Appl. Eng. Educ.*, vol. 21, n.º 1, pp. 135–146, 2013.

[24] B. K. Baradwaj y S. Pal, «Mining educational data to analyze students' performance», *ArXiv Prepr. ArXiv12013417*, 2012.

[25] V. B. Frank, *Classification trees: C4. 5*. 2003.

[26] P. Ajith, M. S. S. Sai, y B. Tejaswi, «Evaluation of Student Performance: An Outlier Detection Perspective», *Int. J. Innov. Technol. Explor. Eng. IJITEE ISSN*, pp. 2278–3075, 2013.

[27] Study.com, «What Is a Decision Tree? - Examples, Advantages & Role in Management - Video & Lesson Transcript», *Study.com*. [En línea]. Disponible en: http://study.com/academy/lesson/what-is-a-decision-tree-examples-advantages-role-in-management.html. [Accedido: 31-oct-2015].

[28] Hortonworks, «Apache Mahout». [En línea]. Disponible en: http://hortonworks.com/hadoop/mahout/. [Accedido: 22-oct-2015].

[29] What - When - How, «Facial Landmark Localization (Face Recognition Techniques) Part 1». [En línea]. Disponible en: http://what-when-how.com/face-recognition/facial-landmark-localization-face-recognition-techniques-part-1/. [Accedido: 04-nov-2015].

[30] G. Julián, «Las redes neuronales: qué son y por qué están volviendo», *Xataka*, diciembre-2014. [En línea]. Disponible en: http://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo. [Accedido: 31-oct-2015].

[31] «Support vector machine», *Wikipedia, the free encyclopedia*. 30-oct-2015.

[32] Trevor Whitney, «Data Mining > Classification». [En línea]. Disponible en: http://trevorwhitney.com/data_mining/classification. [Accedido: 31-oct-2015].

[33] N. Alldrin, A. Smith, y D. Turnbull, «Clustering with EM and K-means», *Univ. San Diego Calif. Tech Rep.*, pp. 261–95, 2003.

[34] PyPR, «K-Means». [En línea]. Disponible en: http://pypr.sourceforge.net/kmeans.html. [Accedido: 31-oct-2015].

[35] «k-means clustering», *Wikipedia, the free encyclopedia*. 18-oct-2015.

[36] O. Maimon y L. Rokach, *Data mining and knowledge discovery handbook*, vol. 2. Springer, 2005.

[37] What is Networking, «Clustering Techniques». [En línea]. Disponible en: http://www.whatisnetworking.net/clustering-techniques/. [Accedido: 01-nov-2015].

[38] R. Agrawal y R. Srikant, «Mining sequential patterns», en *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, 1995, pp. 3–14.

[39] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, y M.-C. Hsu, «Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth», en *icccn*, 2001, p. 0215.

[40] «Educational Data Mining». [En línea]. Disponible en: http://educationaldatamining.org/. [Accedido: 04-nov-2015].

[41] R. S. Baker y K. Yacef, «The state of educational data mining in 2009: A review and future visions», *JEDM-J. Educ. Data Min.*, vol. 1, n.º 1, pp. 3–17, 2009.

[42] C. Romero y S. Ventura, «Educational data mining: a review of the state of the art», *Syst. Man Cybern. Part C Appl. Rev. IEEE Trans. On*, vol. 40, n.º 6, pp. 601–618, 2010.

[43] C. Romero, P. González, S. Ventura, M. J. del Jesús, y F. Herrera, «Evolutionary algorithms for subgroup discovery in e-learning: A practical application using Moodle data», *Expert Syst. Appl.*, vol. 36, n.º 2, pp. 1632–1644, 2009.

[44] M. I. Lopez, J. M. Luna, C. Romero, y S. Ventura, «Classification via Clustering for Predicting Final Marks Based on Student Participation in Forums.», *Int. Educ. Data Min. Soc.*, 2012.

[45] C. Romero, S. Ventura, P. G. Espejo, y C. Hervás, «Data mining algorithms to classify students», en *Educational Data Mining 2008*, 2008.

[46] S. Valsamidis, S. Kontogiannis, I. Kazanidis, y A. Karakos, «E-learning platform usage analysis», *Interdiscip. J. E-Learn. Learn. Objects*, vol. 7, n.º 1, pp. 185–204, 2011.

[47] M. D. Calvo-Flores, E. G. Galindo, M. P. Jiménez, y O. P. Pineiro, «Predicting students' marks from Moodle logs using neural network models», *Curr. Dev. Technol.-Assist. Educ.*, vol. 1, pp. 586–590, 2006.

[48] A. Merceron y K. Yacef, «Interestingness measures for association rules in educational data», en *Educational Data Mining 2008*, 2008.

[49] M. H. Falakmasir y J. Habibi, «Using educational data mining methods to study the impact of virtual classroom in e-learning», en *Educational Data Mining 2010*, 2010.

[50] O. R. Zaíane, «Building a recommender agent for e-learning systems», en *Computers in Education, 2002. Proceedings. International Conference on*, 2002, pp. 55–59.

[51] A. El-Halees, «Mining students data to analyze e-Learning behavior: A Case Study», *Dep. Comput. Sci. Islam. Univ. Gaza PO Box*, vol. 108, 2009.

[52] L. V. Morris, C. Finnegan, y S.-S. Wu, «Tracking student behavior, persistence, and achievement in online courses», *Internet High. Educ.*, vol. 8, n.º 3, pp. 221–231, 2005.

[53] E. Chandra y K. Nandhini, «Knowledge mining from student data», *Eur. J. Sci. Res.*, vol. 47, n.º 1, pp. 156–163, 2010.

[54] C. Vialardi, J. Bravo Agapito, L. S. Shafti, y A. Ortigosa, «Recommendation in higher education using data mining techniques», 2009.

[55] S. Cetintas, L. Si, Y. P. Xin, y C. Hord, «Automatic detection of off-task behaviors in intelligent tutoring systems with machine learning techniques», *Learn. Technol. IEEE Trans. On*, vol. 3, n.º 3, pp. 228–236, 2010.

[56] Apache Software Foundation, «MapReduce Tutorial». [En línea]. Disponible en: http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html. [Accedido: 15-oct-2015].

[57] IBM, «What is MapReduce». [En línea]. Disponible en: http://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/. [Accedido: 15-oct-2015].

[58] M. F. Pace, «BSP vs MapReduce», *Procedia Comput. Sci.*, vol. 9, pp. 246-255, 2012.

[59] C. Dobre y F. Xhafa, «Parallel programming paradigms and frameworks in big data era», *Int. J. Parallel Program.*, vol. 42, n.º 5, pp. 710–738, 2014.

[60] j2kun, «On the Computational Complexity of MapReduce», *Math ∩ Programming*. .

[61] Microsoft, «Dryad - Microsoft Research». [En línea]. Disponible en: http://research.microsoft.com/en-us/projects/dryad/. [Accedido: 15-oct-2015].

[62] Microsoft, «Dryad and DryadLINQ for Data Intensive Research - Microsoft Research». [En línea]. Disponible en: http://research.microsoft.com/en-us/collaboration/tools/dryad.aspx. [Accedido: 15-oct-2015].

[63] Microsoft, «Microsoft drops Dryad; puts its big-data bets on Hadoop». [En línea]. Disponible en: http://www.zdnet.com/article/microsoft-drops-dryad-puts-its-big-data-bets-on-hadoop/. [Accedido: 15-oct-2015].

[64] L. G. Valiant, «A bridging model for parallel computation», *Commun. ACM*, vol. 33, n.º 8, pp. 103–111, 1990.

[65] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, y G. Czajkowski, «Pregel: a system for large-scale graph processing», en *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 135–146.

[66] Apache Software Foundation, «Apache Hama - Big Data and High-Performance Computing». [En línea]. Disponible en: https://hama.apache.org/. [Accedido: 15-oct-2015].

[67] Dato, «Graphlab Create». [En línea]. Disponible en: https://dato.com/products/create/. [Accedido: 15-oct-2015].

[68] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, y J. M. Hellerstein, «Distributed GraphLab: a framework for machine learning and data mining in the cloud», *Proc. VLDB Endow.*, vol. 5, n.º 8, pp. 716–727, 2012.

[69] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, y J. Hellerstein, «Graphlab: A new framework for parallel machine learning», *ArXiv Prepr. ArXiv14082041*, 2014.

[70] G. Mone, «Beyond Hadoop», *Commun. ACM*, vol. 56, n.º 1, pp. 22–24, 2013.

[71] GraphLab Carnegie Mellon, «GraphLab: A New Parallel Framework for Machine Learning». [En línea]. Disponible en: http://www.select.cs.cmu.edu/code/graphlab/. [Accedido: 15-oct-2015].

[72] D. Singh y C. K. Reddy, «A survey on platforms for big data analytics», *J. Big Data*, vol. 2, n.º 1, pp. 1–20, 2015.

[73] DZone.com, «Big Data Analytics Beyond Hadoop». [En línea]. Disponible en: https://dzone.com/articles/big-data-analytics-beyond. [Accedido: 15-oct-2015].

[74] ByteMining.com, «Hadoop Fatigue — Alternatives to Hadoop». [En línea]. Disponible en: http://www.bytemining.com/2011/08/hadoop-fatigue-alternatives-to-hadoop/. [Accedido: 15-oct-2015].

[75] FromDev.com, «35+ Hadoop Alternatives For Big Data». [En línea]. Disponible en: http://www.fromdev.com/2015/03/hadoop-alternatives.html. [Accedido: 15-oct-2015].

[76] S. N. Srirama, P. Jakovits, y E. Vainikko, «Adapting scientific computing problems to clouds using MapReduce», *Future Gener. Comput. Syst.*, vol. 28, n.º 1, pp. 184–192, 2012.

[77] GitHub, «bashreduce · GitHub». [En línea]. Disponible en: https://github.com/erikfrey/bashreduce. [Accedido: 16-oct-2015].

[78] R. Tudoran, A. Costan, y G. Antoniu, «Mapiterativereduce: a framework for reduction-intensive data processing on azure clouds», en *Proceedings of third international workshop on MapReduce and its Applications Date*, 2012, pp. 9–16.

[79] H. Yang, A. Dasdan, R.-L. Hsiao, y D. S. Parker, «Map-reduce-merge: simplified relational data processing on large clusters», en *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 1029–1040.

[80] Disco Project, «Disco MapReduce». [En línea]. Disponible en: http://discoproject.org/. [Accedido: 16-oct-2015].

[81] Google, «Cloud MapReduce – A MapReduce implementation on Amazon Cloud OS - Google Project Hosting». [En línea]. Disponible en: https://code.google.com/p/cloudmapreduce/. [Accedido: 16-oct-2015].

[82] H. Liu y D. Orban, «Cloud mapreduce: A mapreduce implementation on top of a cloud operating system», en *Proceedings of the 2011 11th IEEE/ACM international symposium on cluster, cloud and grid computing*, 2011, pp. 464-474.

[83] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, y G. Fox, «Twister: a runtime for iterative mapreduce», en *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 2010, pp. 810–818.

[84] Twister, «Twister: Iterative MapReduce». [En línea]. Disponible en: http://www.iterativemapreduce.org/. [Accedido: 16-oct-2015].

[85] Google, «HaLoop - An modified version of Hadoop to support efficient iterative data processing on large commodity clusters - Google Project Hosting». [En línea]. Disponible en: https://code.google.com/p/haloop/. [Accedido: 16-oct-2015].

[86] Apache Software Foundation, «Apache Spark™». [En línea]. Disponible en: http://spark.apache.org/. [Accedido: 16-oct-2015].

[87] Apache Software Foundation, «Apache Storm». [En línea]. Disponible en: http://storm.apache.org/. [Accedido: 16-oct-2015].

[88] Hortonworks, «Hortonworks : Open Enterprise Hadoop». [En línea]. Disponible en: http://hortonworks.com/. [Accedido: 16-oct-2015].

[89] Cloudera, «Cloudera». [En línea]. Disponible en: http://www.cloudera.com/content/www/en-us.html. [Accedido: 16-oct-2015].

[90] MapR, «Apache Hadoop Distribution». [En línea]. Disponible en: https://www.mapr.com/. [Accedido: 20-oct-2015].

[91] Network World, «Comparing the top Hadoop distributions». [En línea]. Disponible en: http://www.networkworld.com/article/2369327/software/comparing-the-top-hadoop-distributions.html. [Accedido: 16-oct-2015].

[92] DeZyre.com, «Top 6 Hadoop Vendors providing Big Data Solutions in Open Data Platform». [En línea]. Disponible en: http://www.dezyre.com/article/-top-6-hadoop-vendors-providing-big-data-solutions-in-open-data-platform/93. [Accedido: 16-oct-2015].

[93] Hortonworks, «Impala vs. Hive Performance Benchmark». [En línea]. Disponible en: https://hortonworks.com/blog/impala-vs-hive-performance-benchmark/. [Accedido: 16-oct-2015].

[94] «What distribution should I choose: Cloudera, Hortonworks or MapR? - Quora». [En línea]. Disponible en: https://www.quora.com/What-distribution-should-I-choose-Cloudera-Hortonworks-or-MapR. [Accedido: 09-abr-2016].

[95] Hortonworks, «New Features in HDP 2.3: Open Enterprise Hadoop». [En línea]. Disponible en: http://hortonworks.com/hdp/whats-new/. [Accedido: 18-oct-2015].

[96] Apache Software Foundation, «Apache Hadoop». [En línea]. Disponible en: https://hadoop.apache.org/. [Accedido: 18-oct-2015].

[97] Hortonworks, «Getting Started with HDP - Hortonworks». [En línea]. Disponible en: http://hortonworks.com/hadoop-tutorial/hello-world-an-introduction-to-hadoop-hcatalog-hive-and-pig/. [Accedido: 18-oct-2015].

[98] T. White, *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

[99] J. Dean y S. Ghemawat, «MapReduce: simplified data processing on large clusters», *Commun. ACM*, vol. 51, n.º 1, pp. 107-113, 2008.

[100] Kick Start Hadoop, «Word Count - Hadoop Map Reduce Example», abr-2011. [En línea]. Disponible en: http://kickstarthadoop.blogspot.com.es/2011/04/word-count-hadoop-map-reduce-example.html. [Accedido: 18-oct-2015].

[101] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, y B. Moon, «Parallel data processing with MapReduce: a survey», *AcM SIGMoD Rec.*, vol. 40, n.º 4, pp. 11-20, 2012.

[102] Hortonworks, «Apache Hadoop YARN – Background and an Overview - Hortonworks». [En línea]. Disponible en: http://hortonworks.com/blog/apache-hadoop-yarn-background-and-an-overview/. [Accedido: 19-oct-2015].

[103] Hortonworks, «MapReduce - What MapReduce Does». [En línea]. Disponible en: http://hortonworks.com/hadoop/mapreduce/#section_1. [Accedido: 18-oct-2015].

[104] Yahoo, «Scaling Hadoop to 4000 nodes at Yahoo! | hadoopnew - Yahoo». [En línea]. Disponible en: https://developer.yahoo.com/blogs/hadoop/scaling-hadoop-4000-nodes-yahoo-410.html. [Accedido: 18-oct-2015].

[105] Hortonworks, «Apache Hadoop YARN – Concepts and Applications - Hortonworks». [En línea]. Disponible en: http://hortonworks.com/blog/apache-hadoop-yarn-concepts-and-applications/. [Accedido: 19-oct-2015].

[106] Hortonworks, «What is Hadoop?» [En línea]. Disponible en: http://hortonworks.com/hadoop/. [Accedido: 19-oct-2015].

[107] Apache Software Foundation, «Apache Accumulo». [En línea]. Disponible en: http://accumulo.apache.org/. [Accedido: 19-oct-2015].

[108] Hortonworks, «Apache Accumulo - What Accumulo Does». [En línea]. Disponible en: http://hortonworks.com/hadoop/accumulo/#section_1. [Accedido: 19-oct-2015].

[109] Apache Software Foundation, «Apache HBase». [En línea]. Disponible en: http://hbase.apache.org/. [Accedido: 21-oct-2015].

[110] Hortonworks, «Apache HBase». [En línea]. Disponible en: http://hortonworks.com/hadoop/hbase/#section_1. [Accedido: 21-oct-2015].

[111] Apache Software Foundation, «Apache Kafka». [En línea]. Disponible en: http://kafka.apache.org/. [Accedido: 21-oct-2015].

[112] Hortonworks, «Apache Kafka». [En línea]. Disponible en: http://hortonworks.com/hadoop/kafka/#section_2. [Accedido: 21-oct-2015].

[113] Apache Software Foundation, «Apache Slider». [En línea]. Disponible en: https://slider.incubator.apache.org/. [Accedido: 21-oct-2015].

[114] Hortonworks, «Apache Slider». [En línea]. Disponible en: http://hortonworks.com/hadoop/slider/#section_1. [Accedido: 21-oct-2015].

[115] Hortonworks, «Apache Storm in Hadoop». [En línea]. Disponible en: http://hortonworks.com/hadoop/storm/#section_2. [Accedido: 21-oct-2015].

[116] Apache Software Foundation, «Apache Solr». [En línea]. Disponible en: http://lucene.apache.org/solr/. [Accedido: 21-oct-2015].

[117] Hortonworks, «Apache Solr». [En línea]. Disponible en: http://hortonworks.com/hadoop/solr/. [Accedido: 21-oct-2015].

[118] Apache Software Foundation, «Apache Atlas – Data Governance and Metadata framework for Hadoop». [En línea]. Disponible en: http://atlas.incubator.apache.org/. [Accedido: 21-oct-2015].

[119] Hortonworks, «Apache Atlas». [En línea]. Disponible en: http://hortonworks.com/hadoop/atlas/. [Accedido: 21-oct-2015].

[120] Apache Software Foundation, «Falcon - Falcon - Feed management and data processing platform». [En línea]. Disponible en: http://falcon.apache.org/. [Accedido: 21-oct-2015].

[121] Hortonworks, «Apache Falcon». [En línea]. Disponible en: http://hortonworks.com/hadoop/falcon/. [Accedido: 21-oct-2015].

[122] Apache Software Foundation, «Welcome to Apache Flume — Apache Flume». [En línea]. Disponible en: https://flume.apache.org/. [Accedido: 21-oct-2015].

[123] Hortonworks, «Apache Flume». [En línea]. Disponible en: http://hortonworks.com/hadoop/flume/. [Accedido: 21-oct-2015].

[124] Apache Software Foundation, «Knox Gateway – REST API Gateway for the Hadoop Ecosystem». [En línea]. Disponible en: https://knox.apache.org/. [Accedido: 21-oct-2015].

[125] Hortonworks, «Apache Knox Gateway - Security for Hadoop». [En línea]. Disponible en: http://hortonworks.com/hadoop/knox-gateway/. [Accedido: 21-oct-2015].

[126] Apache Software Foundation, «Apache Ranger - Introduction». [En línea]. Disponible en: http://ranger.incubator.apache.org/. [Accedido: 21-oct-2015].

[127] Hortonworks, «Apache Ranger». [En línea]. Disponible en: http://hortonworks.com/hadoop/ranger/. [Accedido: 21-oct-2015].

[128] Hortonworks, «HDFS». [En línea]. Disponible en: http://hortonworks.com/hadoop/hdfs/#section_2. [Accedido: 19-oct-2015].

[129] Apache Software Foundation, «Apache Hive TM». [En línea]. Disponible en: https://hive.apache.org/. [Accedido: 19-oct-2015].

[130] Apache Software Foundation, «Home - Apache Hive». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/Home. [Accedido: 19-oct-2015].

[131] Hortonworks, «Apache Hive & Hadoop - What Hive Does». [En línea]. Disponible en: http://hortonworks.com/hadoop/hive/#section_1. [Accedido: 19-oct-2015].

[132] Apache Software Foundation, «HCatalog Using HCatalog - Apache Hive». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/HCatalog+UsingHCat. [Accedido: 20-oct-2015].

[133] Hortonworks, «Apache Hive & Hadoop - HCatalog». [En línea]. Disponible en: http://hortonworks.com/hadoop/hive/#section_4. [Accedido: 20-oct-2015].

[134] Apache Software Foundation, «SerDe - Apache Hive». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/SerDe. [Accedido: 20-oct-2015].

[135] Apache Software Foundation, «WebHCat Using WebHCatalog - Apache Hive». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/WebHCat+UsingWebHCat. [Accedido: 20-oct-2015].

[136] Apache Software Foundation, «HCatalog CLI - Apache Hive». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/HCatalog+CLI#HCatalogCLI-HCatalogDDL. [Accedido: 20-oct-2015].

[137] Apache Software Foundation, «Hive Web Interface - Apache Hive». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/HiveWebInterface. [Accedido: 20-oct-2015].

[138] Apache Software Foundation, «Language Manual - Apache Hive». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/LanguageManual. [Accedido: 20-oct-2015].

[139] Hortonworks, «Cheat Sheet: Hive for SQL Users». [En línea]. Disponible en: http://hortonworks.com/wp-content/uploads/2013/05/hql_cheat_sheet.pdf. [Accedido: 20-oct-2015].

[140] Apache Software Foundation, «Welcome to Apache Pig!» [En línea]. Disponible en: https://pig.apache.org/. [Accedido: 21-oct-2015].

[141] Hortonworks, «Apache Pig». [En línea]. Disponible en: http://hortonworks.com/hadoop/pig/. [Accedido: 21-oct-2015].

[142] Hortonworks, «Apache Pig - How Pig Works». [En línea]. Disponible en: http://hortonworks.com/hadoop/pig/#section_2. [Accedido: 21-oct-2015].

[143] Apache Software Foundation, «Pig - Getting Started». [En línea]. Disponible en: http://pig.apache.org/docs/r0.15.0/start.html. [Accedido: 21-oct-2015].

[144] Apache Software Foundation, «Pig Latin Basics». [En línea]. Disponible en: http://pig.apache.org/docs/r0.15.0/basic.html. [Accedido: 21-oct-2015].

[145] Apache Software Foundation, «Pig - Built In Functions». [En línea]. Disponible en: http://pig.apache.org/docs/r0.15.0/func.html. [Accedido: 21-oct-2015].

[146] Apache Software Foundation, «Pig - User Defined Functions». [En línea]. Disponible en: http://pig.apache.org/docs/r0.15.0/udf.html. [Accedido: 21-oct-2015].

[147] Apache Software Foundation, «PiggyBank - Apache Pig». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/PIG/PiggyBank. [Accedido: 21-oct-2015].

[148] Apache Software Foundation, «PigTools - Apache Pig». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/PIG/PigTools. [Accedido: 21-oct-2015].

[149] Apache Software Foundation, «Pig - Control Structures». [En línea]. Disponible en: http://pig.apache.org/docs/r0.15.0/cont.html. [Accedido: 21-oct-2015].

[150] Cloudera, «Managing Spark». [En línea]. Disponible en: http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/admin_spark.html. [Accedido: 21-oct-2015].

[151] Hortonworks, «Spark - In-Memory Compute for Data Science». [En línea]. Disponible en: http://hortonworks.com/hadoop/spark/#section_1. [Accedido: 21-oct-2015].

[152] Apache Software Foundation, «Spark SQL & DataFrames | Apache Spark». [En línea]. Disponible en: http://spark.apache.org/sql/. [Accedido: 21-oct-2015].

[153] Apache Software Foundation, «MLlib | Apache Spark». [En línea]. Disponible en: http://spark.apache.org/mllib/. [Accedido: 21-oct-2015].

[154] Apache Software Foundation, «GraphX | Apache Spark». [En línea]. Disponible en: http://spark.apache.org/graphx/. [Accedido: 21-oct-2015].

[155] Apache Software Foundation, «Spark Streaming | Apache Spark». [En línea]. Disponible en: http://spark.apache.org/streaming/. [Accedido: 21-oct-2015].

[156] Apache Software Foundation, «Cluster Mode Overview - Spark 1.5.1 Documentation». [En línea]. Disponible en: http://spark.apache.org/docs/latest/cluster-overview.html. [Accedido: 21-oct-2015].

[157] Apache Software Foundation, «Spark Standalone Mode - Spark 1.5.1 Documentation». [En línea]. Disponible en: http://spark.apache.org/docs/latest/spark-standalone.html. [Accedido: 21-oct-2015].

[158] Cloudera, «Running Spark Applications on YARN». [En línea]. Disponible en: http://www.cloudera.com/content/www/en-

us/documentation/enterprise/latest/topics/cdh_ig_running_spark_on_yarn.html#concept_asc_2hr_gs_unique_1. [Accedido: 21-oct-2015].

[159] Cloudera, «Running Spark Applications». [En línea]. Disponible en: http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/cdh_ig_running_spark_apps.html. [Accedido: 21-oct-2015].

[160] BigDataHispano, «Apache Spark, la nueva estrella de Big Data». [En línea]. Disponible en: http://www.bigdatahispano.org/noticias/apache-spark-la-nueva-estrella-de-big-data/. [Accedido: 21-oct-2015].

[161] Quora, «What is the difference between Apache Spark and Apache Hadoop (Map-Reduce)». [En línea]. Disponible en: https://www.quora.com/What-is-the-difference-between-Apache-Spark-and-Apache-Hadoop-Map-Reduce. [Accedido: 21-oct-2015].

[162] The Next Platform, «Flink Sparks Next Wave of Distributed Data Processing», *The Next Platform*. [En línea]. Disponible en: http://www.nextplatform.com/2015/02/22/flink-sparks-next-wave-of-distributed-data-processing/. [Accedido: 04-nov-2015].

[163] Apache Software Foundation, «MLlib - Spark 1.5.1 Documentation». [En línea]. Disponible en: http://spark.apache.org/docs/latest/mllib-guide.html. [Accedido: 22-oct-2015].

[164] Apache Software Foundation, «Apache Mahout: Scalable machine learning and data mining». [En línea]. Disponible en: http://mahout.apache.org/. [Accedido: 22-oct-2015].

[165] Apache Software Foundation, «Apache Mahout: Algorithms». [En línea]. Disponible en: https://mahout.apache.org/users/basics/algorithms.html. [Accedido: 22-oct-2015].

[166] Apache Software Foundation, «Apache Tez – Welcome to Apache Tez». [En línea]. Disponible en: https://tez.apache.org/. [Accedido: 21-oct-2015].

[167] Hortonworks, «Apache Tez». [En línea]. Disponible en: http://hortonworks.com/hadoop/tez/#section_1. [Accedido: 21-oct-2015].

[168] InfoQ, «What is Apache Tez?» [En línea]. Disponible en: http://www.infoq.com/articles/apache-tez-saha-murthy. [Accedido: 21-oct-2015].

[169] Hortonworks, «Data Processing API in Apache Tez - Hortonworks». [En línea]. Disponible en: http://hortonworks.com/blog/expressing-data-processing-in-apache-tez/. [Accedido: 21-oct-2015].

[170] Dong, «Apache Tez». .

[171] Apache Software Foundation, «Apache Sqoop». [En línea]. Disponible en: http://sqoop.apache.org/. [Accedido: 23-oct-2015].

[172] Hortonworks, «Apache Sqoop - What Sqoop Does». [En línea]. Disponible en: http://hortonworks.com/hadoop/sqoop/#section_1. [Accedido: 23-oct-2015].

[173] Divakar BigData - Blogspot, «Big Data/Hadoop: Apache Sqoop -Part 1: Basic Concepts». [En línea]. Disponible en: http://divakarbigdata.blogspot.com.es/2014/02/sqoop-concepts-and-installation.html. [Accedido: 23-oct-2015].

[174] Apache Software Foundation, «Apache Sqoop - How Sqoop Works». [En línea]. Disponible en: http://hortonworks.com/hadoop/sqoop/#section_2. [Accedido: 23-oct-2015].

[175] Apache Software Foundation, «Sqoop User Guide (v1.4.6)». [En línea]. Disponible en: http://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html. [Accedido: 23-oct-2015].

[176] Cloudera Engineering Blog, «Apache Sqoop – Overview». [En línea]. Disponible en: http://blog.cloudera.com/blog/2011/10/apache-sqoop-overview/. [Accedido: 23-oct-2015].

[177] Apache Software Foundation, «Oozie - Apache Oozie Workflow Scheduler for Hadoop». [En línea]. Disponible en: http://oozie.apache.org/. [Accedido: 23-oct-2015].

[178] Hortonworks, «Apache Oozie - What Oozie Does». [En línea]. Disponible en: http://hortonworks.com/hadoop/oozie/#section_1. [Accedido: 23-oct-2015].

[179] Hortonworks, «Apache Oozie - How Oozie Works». [En línea]. Disponible en: http://hortonworks.com/hadoop/oozie/#section_2. [Accedido: 23-oct-2015].

[180] Apache Software Foundation, «Oozie Documentation - Overview». [En línea]. Disponible en: http://oozie.apache.org/docs/4.2.0/DG_Overview.html. [Accedido: 23-oct-2015].

[181] CtrlAGeeks, «Apache Oozie Introduction». [En línea]. Disponible en: http://www.ctrlageeks.com/apache-oozie-introduction/. [Accedido: 23-oct-2015].

[182] Apache Software Foundation, «Apache Ambari». [En línea]. Disponible en: https://ambari.apache.org/. [Accedido: 23-oct-2015].

[183] Apache Software Foundation, «Apache Ambari - What Ambari Does». [En línea]. Disponible en: http://hortonworks.com/hadoop/ambari/#section_1. [Accedido: 23-oct-2015].

[184] Apache Software Foundation, «Apache Ambari - Ambari User Views». [En línea]. Disponible en: http://hortonworks.com/hadoop/ambari/#section_4. [Accedido: 23-oct-2015].

[185] Hue, «Hue - Hadoop User Experience - The Apache Hadoop UI | Hue is a Web application for querying and visualizing data by interacting with Apache Hadoop». [En línea]. Disponible en: http://gethue.com/. [Accedido: 23-oct-2015].

[186] Apache Software Foundation, «Apache ZooKeeper - Home». [En línea]. Disponible en: https://zookeeper.apache.org/. [Accedido: 21-oct-2015].

[187] Hortonworks, «Apache ZooKeeper». [En línea]. Disponible en: http://hortonworks.com/hadoop/zookeeper/. [Accedido: 21-oct-2015].

[188] elastic, «Elasticsearch: RESTful, Distributed Search & Analytics | Elastic». [En línea]. Disponible en: https://www.elastic.co/products/elasticsearch. [Accedido: 24-oct-2015].

[189] elastic, «Elasticsearch Reference [1.7]». [En línea]. Disponible en: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html. [Accedido: 24-oct-2015].

[190] Eonic, «Cluster configuration». [En línea]. Disponible en: https://enonic.com/docs/4.7/cluster-configuration.html. [Accedido: 24-oct-2015].

[191] EverythingShouldBeVirtual, «Highly Available ELK (Elasticsearch, Logstash and Kibana) Setup». [En línea]. Disponible en: http://everythingshouldbevirtual.com/highly-available-elk-elasticsearch-logstash-kibana-setup. [Accedido: 04-nov-2015].

[192] elastic, «Hadoop: Immediate Insight into Big Data | Elastic». [En línea]. Disponible en: https://www.elastic.co/products/hadoop. [Accedido: 24-oct-2015].

[193] elastic, «Kibana: Explore, Visualize, Discover Data | Elastic». [En línea]. Disponible en: https://www.elastic.co/products/kibana. [Accedido: 24-oct-2015].

[194] elastic, «Kibana: Introduction». [En línea]. Disponible en: https://www.elastic.co/guide/en/kibana/current/introduction.html. [Accedido: 24-oct-2015].

[195] Hue Team, «Hue 3 on HDP installation tutorial – Hue - Hadoop User Experience...», *Hue – Hadoop User Experience – The Apache Hadoop UI*, 12-feb-2015. .

[196] Siva, «Hive Performance Tuning», *Hadoop Online Tutorials*, 03-may-2015. .

[197] «Elastic · Revealing Insights from Data (Formerly Elasticsearch)». [En línea]. Disponible en: https://www.elastic.co/. [Accedido: 08-mar-2016].

[198] «LanguageManual Types - Apache Hive - Apache Software Foundation». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types. [Accedido: 25-feb-2016].

[199] «LanguageManual UDF - Apache Hive - Apache Software Foundation». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-Built-inAggregateFunctions(UDAF). [Accedido: 25-feb-2016].

[200] «LanguageManual LateralView - Apache Hive - Apache Software Foundation». [En línea]. Disponible en: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView. [Accedido: 26-feb-2016].

[201] Siva, «Partitioning in Hive», *Hadoop Online Tutorials*, 14-dic-2014. .

[202] E. Capriolo, D. Wampler, y J. Rutherglen, *Programming Hive*. O'Reilly Media, Inc., 2012.

[203] Siva, «Bucketing In Hive», *Hadoop Online Tutorials*, 20-dic-2014. .

[204] Moodle, «Activities - MoodleDocs». [En línea]. Disponible en: https://docs.moodle.org/23/en/Activities. [Accedido: 25-feb-2016].

[205] «Pearson product-moment correlation coefficient», *Wikipedia, the free encyclopedia*. 01-feb-2016.

[206] S. Owen, R. Anil, T. Dunning, y E. Friedman, *Mahout in action*. Manning Shelter Island, 2011.

[207] Daniel Nydegger, «Mahout Clustering Example - Software Development». .

[208] Sujit Pal, «Salmon Run: Search Rules using Mahout's Association Rule Mining». .

[209] Olivia Klose, «Mahout for Dummies (3) – Step-by-Step: Mahout with HDInsight PowerShell Style - Olivia's Blog - Site Home - TechNet Blogs». .

[210] BigData Explorer, «Apache Mahout Random Forest | Wei Shung Chung». [En línea]. Disponible en: http://weishungchung.com/2014/10/10/apache-mahout-random-forest/. [Accedido: 25-mar-2016].

[211] Mark Needham, «Kaggle Digit Recognizer: Mahout Random Forest attempt», *Mark Needham*, 27-oct-2012. .

[212] Mark Needham, «Mahout: Using a Saved Random Forest/DecisionTree - DZone Big Data», *dzone.com*. [En línea]. Disponible en: https://dzone.com/articles/mahout-using-saved-random. [Accedido: 25-mar-2016].

[213] M. K. Islam y A. Srinivasan, *Apache Oozie: The Workflow Scheduler for Hadoop*. O'Reilly Media, Inc., 2015.

[214] «Oozie - Workflow Functional Specification». [En línea]. Disponible en: https://oozie.apache.org/docs/4.2.0/WorkflowFunctionalSpec.html. [Accedido: 03-mar-2016].

[215] «Oozie - Coordinator Functional Specification». [En línea]. Disponible en: https://oozie.apache.org/docs/4.2.0/CoordinatorFunctionalSpec.html. [Accedido: 03-mar-2016].

[216] «Oozie - Coordinator Functional Specification - DataSet». [En línea]. Disponible en: https://oozie.apache.org/docs/4.2.0/CoordinatorFunctionalSpec.html#a5._Dataset. [Accedido: 03-mar-2016].

[217] «Ooize Data Pipeline Done-Flag». [En línea]. Disponible en: http://nathan.vertile.com/blog/2014/09/02/oozie-data-pipeline-done-flag/#top. [Accedido: 12-mar-2016].

[218] «Oozie - Bundle Functional Specification». [En línea]. Disponible en: https://oozie.apache.org/docs/4.2.0/BundleFunctionalSpec.html. [Accedido: 03-mar-2016].

[219] N. Barcelo, N. Legg, y T. Bressoud, «The performance cost of virtual machines on big data problems in compute clusters», en *Proceedings of the Midstates Conference for Undergraduate Research in Computer Science and Mathematics*, 2008, pp. 22–29.

[220] T. Ivanov, R. V. Zicari, S. Izberovic, y K. Tolle, «Performance Evaluation of Virtualized Hadoop Clusters», *ArXiv Prepr. ArXiv14113811*, 2014.

[221] M. Ishii, J. Han, y H. Makino, «Design and performance evaluation for hadoop clusters on virtualized environment», en *Information Networking (ICOIN), 2013 International Conference on*, 2013, pp. 244–249.

[222] «Performance Comparison of Big Data Analysis using Hadoop in Physical and Virtual Servers». [En línea]. Disponible en: http://www.cs.wustl.edu/~jain/cse570-13/ftp/bigdatap/index.html#sec4. [Accedido: 10-mar-2016].

[223] VMware, Inc., «A Benchmarking Case Study of Virtualized Hadoop Performance on vSphere 5 - White Paper: VMware, Inc. - VMW-Hadoop-Performance-vSphere5.pdf». [En línea]. Disponible en: https://www.vmware.com/files/pdf/VMW-Hadoop-Performance-vSphere5.pdf. [Accedido: 10-mar-2016].

Contenido del CD adjunto

- Memoria del trabajo en formato *PDF*.
- Código fuente del trabajo ubicado en el directorio "*CódigoFuente"*, tal y como se indicó en el apartado 4.4.