The Journal of Supercomputing (2021) 77:9934–9946
https://doi.org/10.1007/s11227-021-03667-x

A methodology to enable QoS provision on InfiniBand hardware

Javier Cano-Cano[1] · **Francisco J. Andújar**[2] · **Jesús Escudero-Sahuquillo**[1] · **Francisco J. Alfaro-Cortés**[1] · **José L. Sánchez**[1]

Accepted: 2 February 2021 / Published online: 22 February 2021

Abstract
Modern high-speed interconnection networks include support for the provision of quality of service (QoS) to the applications. The output scheduling algorithm plays an important role in the QoS provision, choosing the packets to be delivered from the output buffers. InfiniBand, one of the most used interconnection technologies, includes a table-based scheduler composed of a high- and a low-priority tables, and a counter limiting the number of high priority traffic flows that may be delivered before giving the opportunity to low priority ones. Therefore, the performance of the traffic flows in the network largely depends on the table configuration since the switch scheduler uses this information to allow/deny packets being forwarded, according to the QoS provision scheme. As far as we know, there is no study on the influence of these configurations to the traffic flows performance. In this paper, we present an offline analysis tool to accurately determine the expected end-to-end latency and bandwidth of the traffic flows in an InfiniBand-based network using the information contained in the high- and low-priority tables. Moreover, we present a methodology to aid network administrators in configuring the QoS provision in a real InfiniBand cluster. Finally, we evaluate the analysis tool, comparing its results with those obtained from a real cluster and from simulation.

Keywords Analysis tool · High-performance networks · Quality of service · InfiniBand · Performance evaluation · Scheduling algorithms

✉ Javier Cano-Cano
javier.cano@uclm.es

1 Computing System Department, Universidad de Castilla-La Mancha, Albacete, Spain

2 Computing System Department, Universidad de Valladolid, Valladolid, Spain

1 Introduction

High-performance computing (HPC) is expected to break the exascale barrier soon [11]. In HPC systems, the interconnection network is a crucial element, as it needs to process communications generated by HPC applications and, therefore, it may become the entire system bottleneck. In order to overcome this potential problem, many improvements have been devised in the last years, focused on routing algorithms, congestion avoidance mechanisms, switch architectures, etc. [1, 25].

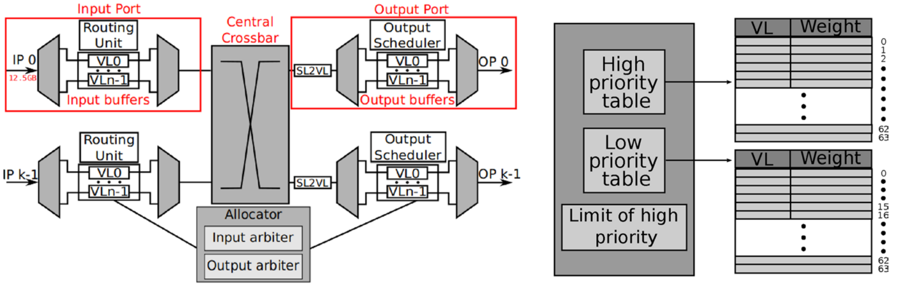
There are multiple HPC interconnection network technologies that have been competing to achieve better performance and market share. Some of the most popular interconnection network technologies are Gigabit Ethernet (GE) [20], InfiniBand (IB) [16] and Omni-Path (OPA) [4]. In the most powerful computer list TOP500 [23], GE has 50.8%, IB has 31% and OPA has 9.4% of market share (November, 2020). But in the top 100, the amount of IB-based supercomputers grows to 61%. As can be seen, IB is one of the most used interconnection technologies by the most powerful HPC systems in the world.

One crucial feature in HPC interconnection networks is the quality of service (QoS) provision. QoS allows to differentiate traffic flows from different applications according to their specific requirements (e.g., latency or throughput). If this functionality is available, network administrators need to configure the QoS provision a priori, avoiding that one or a few traffic flows consume all the network resources, or that the end, users may experience a poor system performance, even if the system is not overloaded. Note that the most popular interconnection network technologies include support for providing QoS, being nowadays an active research topic in HPC environments [18, 19, 22].

The cornerstone of QoS provision is the output scheduling algorithm [7, 8], which determines when a packet should be delivered to the next network device (i.e., network interface, switch, etc.). The decision is based on performance metrics such as end-to-end delay and/or bandwidth requirements. In the context of HPC interconnection networks, this scheduler has to be as simple as possible in terms of computational and implementation complexity [21]. The algorithm latency must be smaller than the transmission time. Otherwise, the system will expend more time choosing the next packet to be delivered than transmitting packets. Also, the implementation complexity must be low because these schedulers are typically implemented in hardware.

A well-known family of scheduling algorithms is table-based schedulers. In fact, both IB and OPA HPC interconnection networks include table-based schedulers [4, 10]. This family of schedulers offers a good latency and bandwidth performance with a low computational complexity. These schedulers have been widely studied over the last few decades [2, 12, 13], especially on IB.

The IB scheduler is based on two arbitration tables: high-priority table and low-priority table. The scheduler also includes a high-priority counter which determines the maximum amount of data that may be transmitted using the high-priority table before giving an opportunity to the low-priority table. The



(a) IB-based k-port switch. **(b)** IB output scheduler.

Fig. 1 Diagram of IB architecture

high-priority table has been widely studied [2, 3, 6]. However, how the low-priority table and the high-priority counter influence the final traffic flows performance, as far as we know, has not yet been analyzed.

In this paper, we present an offline analysis tool that allows users to extract the expected end-to-end latency and bandwidth values of the traffic flows from a given IB scheduler configuration. We also present a methodology to test these QoS configurations in a real IB testbed, and a comparison between the expected values obtained from our offline analysis tool and the achieved results from the testbed.

The structure of the paper is as follows: Sect. 2 describes the QoS support on IB. Section 3 introduces the offline InfiniBand analysis tool and Sect. 4 presents the methodology to test the scheduler configurations in a real IB testbed. Section 5 shows the experimental results. Finally, in Sect. 6, some conclusions are drawn.

2 InfiniBand QoS support

This section reviews the mechanisms provided by the IB architecture to provide QoS to applications. These mechanisms are the virtual lanes (VLs), service levels (SLs), output scheduler and SL2VL tables [6, 10, 16].

VLs provide dedicated buffer space for packets at the input/output ports. Moreover, IB implements a credit-based flow control at VL-level. IB supports up to 16 VLs being the last VL (VL15) reserved for network control traffic.

SLs are the only QoS information stored in packets. SLs provide QoS to the applications aggregating traffic flows with similar characteristics through the network. The packet SL is set by the network interfaces (NICs) before their injection in the network and cannot be modified later.

Moreover, in order to provide QoS, IB switches also require in each output port, the SL-to-VL mapping tables (SL2VL), as shown in Fig. 1a. Through these tables, the packets are assigned to a VL based on their SL, output port and input port. Note that as each output port has its own SL2VL table, packets may be assigned to different VLs along their route.

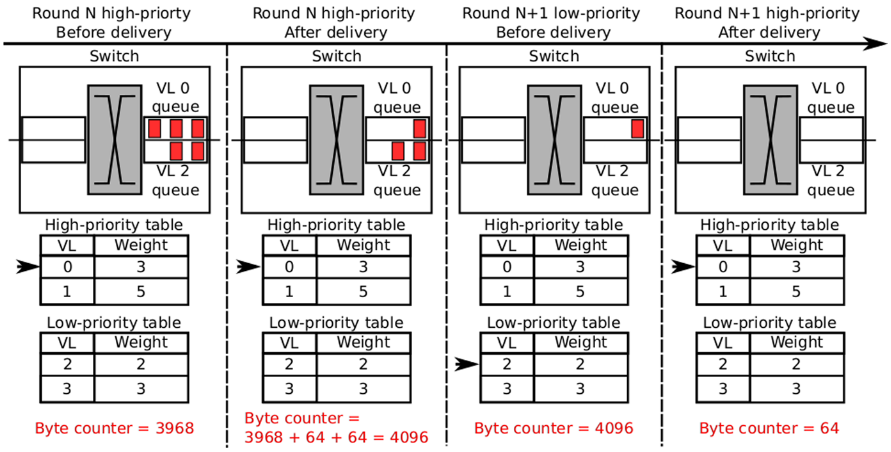


Fig. 2 Example of high-priority and low-priority tables

Figure 1a also shows an output scheduler per output port. This scheduler is a table-based scheduler with two tables, as shown in Fig. 1b. The first table (*High-priority table*) is aimed to schedule high-priority VLs, while the second table (*Low-priority table*) is devoted to low-priority VLs. Both arbitration tables have up to 64 entries. Each entry has a VL identifier and a weight ranging from 0 to 255. The weight indicates how many 64-byte units may be delivered from that VL. Entry weights are always rounded up in order to allow the transmission of an entire packet.

IB output scheduler also defines the *LimitOfHighPriority* counter. It represents the maximum amount of information that may be delivered from the high-priority VLs before selecting the low-priority ones. A byte counter accumulates the amount of information sent by high-priority VLs. When this counter reaches *LimitOfHighPriority* $\times$ 4096 bytes, the scheduler can choose a low-priority VL. If there are no packets in high-priority VLs, the scheduler can also choose a low-priority entry. There are two special values for *LimitOfHighPriority*: 255 indicates that there is no byte limit for high-priority VLs, while 0 indicates that only one high-priority packet can be transmitted, allowing to send packets from both tables alternately.

For instance, let us consider the output IB port shown in Fig. 2. The port has four VLs and one buffer per VL. The high-priority VL 0 has three packets, the low-priority VL 2 has two packets and packets are 64 bytes sized. Finally, *LimitOfHighPriority* = 1 (i.e., 4096 bytes), and the byte counter is 3968.

When the scheduling is needed, the high-priority table is cycled through until an active VL[1] is found. Then the VL can transmit as many packets as defined in the "weight" field of the selected table entry. In the example, the scheduler selects the first table entry, allowing VL 0 (black arrow on the left) to deliver 196 bytes, i.e., three packets (*Round N high-priority Before delivery*). However, the byte counter

[1] A VL is active when it stores packets and has credits to send at least one packet.

reaches 4096 after transmitting the second packet. Given that the VL 0 interrupts the transmission, giving the opportunity to send packets to the low-priority VLs (*Round N high-priority After delivery*). Then, the first low-priority table entry is selected, allowing VL 2 to deliver its two packets (*Round N+1 low-priority Before delivery*). Finally, the byte counter is reset, the delivery of high-priority VLs is resumed and VL 0 sends its last packet.

In a table-based scheduler that only uses a single table, e.g., using the IB output scheduler with only the high-priority table, using a proper table entry weights and distances distribution, the total amount of bandwidth ϕ_i assigned to the VL_i in an N-entry arbitration table is

$$\phi_i = \left(\sum_{j=0}^{J-1} weight_j \right) \Big/ \left(\sum_{k=0}^{N-1} weight_k \right) \quad (1)$$

where J is the amount of table entries assigned to the VL_i, *weight* is the entry weight assigned to a table entry and N is the total number of entries of the arbitration table. However, as far as we know, there is no tool or methodology that could be used by system administrators and researchers to figure out the ϕ_i assigned to the VL_i when the two IB arbitration tables are used, apart from testing the configuration on a real IB cluster or simulator with IB support. Furthermore, the maximum distance is very complicated to know in advance, since it is not possible to know exactly when the byte counter will exceed the *LimitOfHighPriority* and so to interleave a low-priority table entry between two high-priority table entries.

3 Offline InfiniBand multi-table analysis tool

Given a high priority arbitration table configuration, we can easily calculate the bandwidth assigned to each traffic flow and the maximum entry distance between any pair of table entries assigned to the same traffic flow. Intuitively, let's consider that the VL 0 has 32 entries assigned in the high priority table with a distance of 2 between any pair of consecutive entries, the VL 1 has 16 table entries with a distance of 4 entries, and the weight of each VL 0 entry is 2 units while the weight of each VL 1 entry is 4 units. Hence, the total amount of bandwidth assigned is the same (i.e., 50%) for each VL. However, with this configuration, the VL 0 packets will show lower end-to-end latency than that of VL 1 packets, since the distance between table entries is lower for VL 0. Note that, when the *LimitOfHighPriority* is added to the arbitration process, these values become more complicated to obtain. For instance, if a low priority entry and a *LimitOfHighPriority* counter are added to the above example, the bandwidth portion assigned and entry distances cannot be directly obtained. Therefore, predicting in advance the exact moment when the low priority table has the opportunity to deliver packets is not a trivial task.

For this purpose, we have developed an offline analysis tool. The main goal of this tool is to obtain, from a given IB output scheduler configuration, the VL effective bandwidth and maximum distance between any pair of consecutive entries assigned to the same VL. The VL effective bandwidth is a valuable metric because different traffic flows have bandwidth requirements that should be met. The maximum distance allows to control the final latency achieved by each VL [2].

This tool has a range of configurable parameters aimed to cover almost all the possible configurations of the IB output scheduling algorithm. These parameters are:

– **High-priority table:** A comma-separated plain text file including on each line a high-priority table entry. This parameter is mandatory.
– **Low-priority table:** A comma-separated plain text file including on each line a low-priority table entry. This parameter is optional.
– **Limit of high-priority:** This value represents the *LimitOfHighPriority* counter, which ranges from 0 to 255. Its default value is 1.
– **Runs:** It specifies the number of high-priority table cycles performed during the analysis. A table cycle is considered completed when the last high-priority table entry is reached and the scheduler returns to the first table entry. It is recommended to set this parameter at least to 30, which is the default value. However, some configurations may require more runs to converge.
– **Packet size:** The packet size in bytes. This value can be fixed to match the maximum transmission unit (MTU), which on IB networks by default is 4096. Also, the packet size is 64, so that $64 \times 64 = 4096$ bytes. The default value is 4096 bytes.

The analysis tool simulates the delivery of packets in the output ports under the following assumptions: (1) There are always packets ready for delivery and (2) There is always space in the next network device to store the transmitted packet. These assumptions give us an ideal scenario for obtaining the theoretical bandwidth divisions and the maximum entry distances. The tool also includes a statistical module to collect and compute the results. Algorithm 1 shows how the tool simulates the packet delivery and collects statistics.

In Algorithm 1, there are several functions involved: *len*() extracts the number of table entries from a given arbitration table, *get_hp_entry*() gets the next table entry and cycles through the table in a round-robin order, *get_weight_from_entry*() returns the associated weight to the given entry, *deliver_all_lp_packets*() simulates the delivery of as many packets as the next low-priority table entry states, *update_stats*() refreshes the current VL statistics and *deliver_hp_packet*() simulates the delivery of one high-priority packet. Note that high-priority packets are sent out one at a time for the current entry, while low-priority packets of the current entry are delivered at the same time. Unlike high-priority packets, there is not a byte counter limiting the amount of information sent by the low-priority table. The scheduler just delivers as many from the low-priority table packets as the selected entry weight allows.

Algorithm 1 Delivery simulation and statistics collection.

```
procedure RUN(hp, lp, limit_hp, runs)
    num_entries ← len(hp)
    limit_hp_counter ← 0
    for i ← num_entries ∗ runs do
        hp_entry ← get_hp_entry(hp)
        entry_weight ← get_weight_from_entry(hp_entry)
        for j ← entry_weight do
            if limit_hp ≠ 255 and limit_hp_counter ≥ limit_hp then
                lp_entry ← get_lp_entry(hp)
                deliver_all_lp_packets(lp_entry)
                vl_stats ← update_stats(lp_entry)
                limit_hp_counter ← 0
            end if
            deliver_hp_packet(hp_entry)
            limit_hp_counter ← limit_hp_counter + packet_size
            vl_stats ← update_stats(hp_entry)
        end for
    end for
    return vl_stats
end procedure
```

Once Algorithm 1 has finished, the bandwidth ϕ_i and the maximum entry distance dst_i are calculated as follows:

$$\phi_i = \frac{packets_i}{\sum_{j=0}^{V-1} packets_j}, \quad dst_i = \frac{accumDst_i}{timesSelected_i}, \tag{2}$$

where $packets_i$ is the total amount of packets delivered by the VL_i and V is the total number of VLs in the system. The $accumDst_i$ is the sum of distances between any pair of consecutive entries assigned to VL_i and $timesSelected_i$ is the sum of times that the entries associated to VL_i have been selected.

4 InfiniBand QoS testing methodology

This section details a simple methodology to test and compare the performance of different configurations of the IB output scheduler. As stated in Sect. 3, to know in advance the bandwidth assigned to every VL_i from a configuration is not possible, unless we use an analysis tool such as the one presented in Sect. 3, a simulation tool or a real IB network.

The main idea behind our methodology is to create multiple traffic flows which will be sent between two nodes interconnected by means of an IB switch. Each traffic flow will try to use the full bandwidth, and therefore, all the traffic flows will be forced to compete for the switch resources. Thus, the output scheduler will distribute the resources (i.e., buffering space, internal links, etc.) among the traffic flows based on their QoS configuration. Figure 3 shows an example using three traffic flows. The methodology defines the following steps for QoS testing:

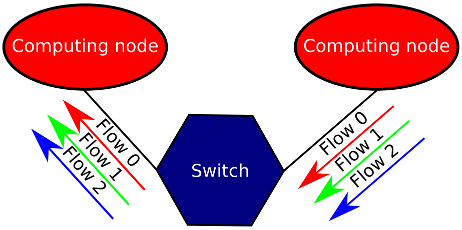


Fig. 3 Traffic flow example inside a IB cluster

1. Define the output scheduler configuration that we want to test. That is, to define the arbitration table configurations (high-priority is mandatory and the low-priority is optional), the *LimitOfHighPriority* counter value (if low-priority table is used) and the SL2VL table configuration.
2. Define the traffic flows. One traffic flow is required for each VL defined on the scheduler configuration in step 1. Each traffic flow is assigned with a different SL at the computing node IB interface, prior to its injection in the network. Each SL is associated with a different VL in the network setup stage through the SL2VL tables, allowing to the output scheduler to segregate the traffic flows and allocate different link bandwidth for each traffic flow.
3. Set up the scheduler configuration (step 1) and the SL2VL tables (step 2) using the subnet manager (SM) [10]. The SM is in charge of multiple actions, such as discovering the topology, configuring the SL2VL and arbitration tables, populating the routing tables, etc. We have used OpenSM [14] which is the common open-source choice on IB networks.
4. Select the source and the destination nodes. The nodes must be interconnected by switches.
5. Generate all the traffic flows simultaneously. Each traffic flow will be assigned to a different SL, according to step 2.
6. Analyze the obtained results. The applications, generally show the benchmark results, which can be stored in a plain text file and later analyzed.

5 Performance analysis

This section compares the results obtained from the offline analysis tool described in Sect. 3 with those achieved from a real IB cluster using the methodology explained in Sect. 4. The main goal of this analysis is to validate the analysis tool behavior.

Moreover, we have carried out experiments using the simulation tool Hiperion [5, 9] to compare the results obtained from the analysis tool against multiple system configurations. The main goal is to find out if the bandwidth obtained by the SLs in the experiments remain accurate when the system size grows.

5.1 Testbed configuration

The IB testbed is composed of one switch Mellanox SB7800 Series 2 EDR 100Gb/s with 36 ports and three computing nodes HPE ProLiant DL380 Gen10 Server. Each

Table 1 Arbitration table configurations

SL	Configuration A			Configuration B		
	SLentries	EntryWeight	SLTotalWeight	SLEntries	EntryWeight	SLTotalWeight
0	32	8–9	264	32	22	704
1	16	9–10	158	16	27	432
2	16	6–7	106	16	18	288
3	1	6	6	1	2	2
Total	65		534	65		1426

node has two Intel Xeon Silver 4116 processors. The operating system is CentOS 8 running OpenSM 3.3.19. To generate the traffic flows, we have used the InfiniBand Perftest package 4.4.0 [15], a collection of software tests written using the IB verbs API [17], which are used as performance micro-benchmarks. Specifically, we have used the `ib_send_bw` and the `ib_send_lat` applications. The `ib_send_bw` is aimed to test the bandwidth sending packages from a sender to a receiver, while `ib_send_lat` application measures the message latency.

We have carried out experiments using two different arbitration table configurations. Table 1 shows both configurations A and B. SL 0, 1 and 2 are high-priority SLs, while SL 3 is a low-priority SL. The *SLEntries* column shows the number of table entries assigned to each SL. For instance, SL 0 has 32 high-priority entries in both configurations and SL 3 has 1 low-priority entry. The *EntryWeight* is the weight assigned to each table entry, while the *SLTotalWeight* column shows the total weight accumulated per each SL. *LimitOfHighPriority* is set to 1 ($1 \times 4096 = 4096$ bytes) in both cases. Note that in configuration A, there are two values in *EntryWeight* column, (e.g., SL 0 has an *EntryWeight* of 8-9). This means that some SL 0 table entries have been assigned with a weight of 8 and others with 9. We have used the methodology proposed at [13] for configuring the high-priority table. However, this methodology may produce that an SL gets more or less bandwidth than desired, so we add or subtract weight from the entries until the bandwidth assigned to each SL is equal or very close to the desired bandwidth. The offline analysis tool is configured using these two configurations, 300 runs and a packet size of 4096 bytes.

5.2 Simulation configurations

We have chosen the k-ary n-tree topology or fat-tree for being very common and well known solution in high performance environments. We have tested this topology with multiple layouts, varying the number of stages ($n \in \{2, 3\}$) and the k-arity ($k \in \{2, 4, 6, 8, 10, 12, 14\}$).

The k-ary n-tree topology implements the valiant routing algorithm, which leverages the available routes in the topology, balancing in a fair manner the traffic flows among them [24]. In these experiments, we have used the Configuration A shown in Table 1. The flit size is 16 bytes and the packet size is 4 flits, resulting in a 64-byte credit unit. The buffer capacity is 7,168 flits at switch input/output ports and 14,336

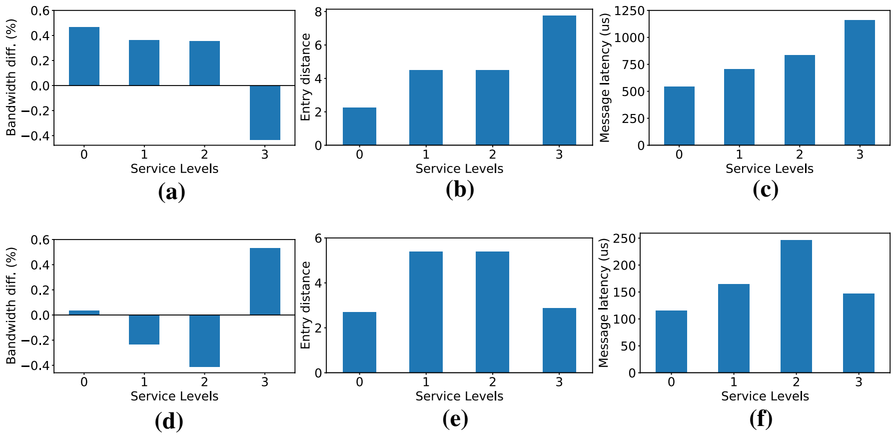


Fig. 4 Performance of each SL using configurations A (**a**, **b**, **c**) and B (**d**, **e**, **f**)

flits at the NICs. Finally, if an application wants to inject a packet into a full NIC queue, we assume that the packet is stored in the application layer queue.

Regarding the traffic, in this study, we have used synthetic traffic. Specifically, we have modeled a random uniform traffic distribution using a fixed generation rate of 1 flit/cycle/NIC. This traffic pattern along with this injection ratio ensures that the network is fully loaded.

5.3 Evaluation results

We have used three metrics in this study:

- **Bandwidth difference:** This is the link bandwidth percentage difference obtained on each system for every SL. This is $\phi_i^{tool} - \phi_i^{IB}$, where ϕ_i^{tool} is the bandwidth percentage obtained by the SL_i with the analysis tool, and ϕ_i^{IB} is the bandwidth percentage obtained by the SL_i on the real system.
- **Maximum entry distance:** It is the maximum distance between any pair of consecutive entries assigned to the same VL. It allows to know the expected VL latency. This metric gives us the end-to-end packet latency.
- **Average message latency:** It measures the end-to-end message latency on the real system.

Note that we have used the message latency instead of the packet latency. In HPC environments, the clock signal of different computing nodes is not synchronized and, there is no global time variable allowing us to know the exact deliver and reception packet timestamps in different nodes. For the same reason, as it is also very complicated the end-to-end message latency we use the *ib_send_lat* application to estimate this metric.

Figure 4 shows the obtained results. Referring bandwidth percentage differences shown in Fig. 4a, d, the differences are always below 0.5%. For instance,

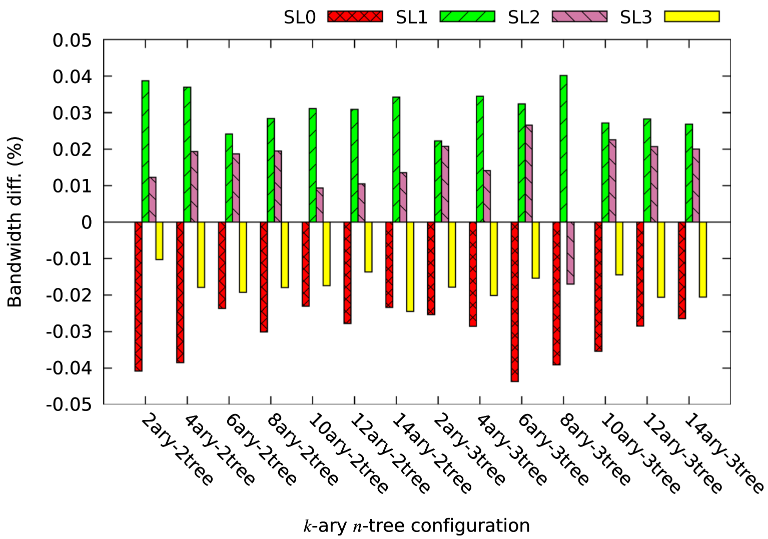


Fig. 5 Bandwidth differences in simulated scenarios (Configuration A)

using the configuration A, the SL 3 gets 8.57% of the total link bandwidth using the analysis tool and 9% in the real system. Hence, the bandwidth difference is $8.57 - 9 = -0.43$ meaning that the SL has achieved 0.43% more than indicated by the analysis tool. These differences are because the IB EDR specification uses an encoding protocol 64b/66b, which means that 2 of 66 delivered bits are used for error control. This may lead to situations where some SLs achieve less bandwidth than specified in the arbitration tables and others allocate those free resources obtaining more bandwidth than specified. Because of it is not possible in advance to know which traffic flows are going to be affected, we have not considered the encoding protocol in our simulation tool in order to obtain a theoretical bandwidth value. Note that, in general, the bandwidth differences are negligible.

Regarding the maximum entry distances and the message latencies shown in Fig. 4b, c, e and f, the final message latency corresponds to the entry distances obtained by the analysis tool. There is an exception in SLs 1 and 2, which theoretically have the same maximum entry distance and on the real system tests the SL 2 achieves more message latency than the SL 1. This is because despite having the same maximum entry distance, the SL 1 has more bandwidth assigned than the SL 2, which indirectly affects to the final message latency as well. However, entry distances results obtained by the analysis tool are useful as a general idea of the expected message latencies.

Figure 5 shows the obtained results for each SL, using the simulated system configurations described in Sect. 5.2. In this case, the main goal is to compare the results of the analysis tool when the system size grows. To carry out this comparison, we have analyzed the bandwidth differences. As shown in Fig. 5, the bandwidth differences remains constant in all tested scenarios and these differences are always below ±0.045%, which is a negligible difference. Therefore, the analysis tool is able to predict the bandwidth division regardless of the system size.

6 Conclusions

In this paper, we have presented an offline InfiniBand multi-table analysis tool capable of extracting from the possible configurations for high and low priority tables; and a *LimitOfHighPriority* counter, the expected end-to-end latency and link bandwidth division. We have also presented a simple methodology to test and validate the same configurations on a real InfiniBand system. Moreover, we have carried out experiments using the simulation tool Hiperion, in order to explore if the analysis tool is accurate when the system size grows. Results show that the analysis tool is able to achieve its objective, in a fast, accurate and simple way. Furthermore, the accuracy of the result obtained does not decrease as the network is scaled. Results show that the analysis tool is capable of extracting the bandwidth division and the maximum entry distance, with negligible differences, in a fast, accurate and simple way.

As future work, we are working on a mathematical approach to configure the IB output scheduler. This approach can determine the optimal scheduler configuration, given a bandwidth and latency requirements, instead of testing multiple configurations looking for the one that meets our requirements.

Acknowledgements This work has been supported by the Junta de Comunidades de Castilla-La Mancha, European Commission (FEDER funds) and Ministerio de Ciencia, Innovación y Universidades under projects SBPLY/17/180501/000498 and RTI2018-098156-B-C52, respectively. It is also co-financed by the University of Castilla-La Mancha and Fondo Europeo de Desarrollo Regional funds under project 2019-GRIN-27060. Javier Cano-Cano is also funded by the MINECO under FPI grant BES-2016-078800.

References

1. Ahn JH, Son YH, Kim J (2013) Scalable high-radix router microarchitecture using a network switch organization. ACM Trans Archit Code Optim (TACO) 10(3):17
2. Alfaro FJ, Sánchez JL, Duato J (2004) QoS in InfiniBand subnetworks. IEEE Trans Paral Distrib Syst 15(9):810–823
3. Alfaro FJ, Sánchez JL, Orozco L, Duato J (2003) Providing QoS in InfiniBand for regular and irregular topologies. In: CCECE 2003-Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology (Cat. No. 03CH37436), vol 2, pp 1079–1082. IEEE
4. Birrittella MS et al (2015) Intel® Omni-Path Architecture: Enabling scalable, high performance fabrics. In: IEEE 23rd Annual Symposium on High-Performance Interconnects (HOTI), 2015, pp 1–9. IEEE
5. Cano-Cano J, Andújar FJ, Alfaro-Cortés FJ, Sánchez JL (2021) QoS provision in hierarchical and non-hierarchical switch architectures. J Paral Distrib Comput 148:138–150
6. Crupnicoff D, Das S, Zahavi E (2005) Deploying quality of service and congestion control in InfiniBand-based data center networks. Mellanox Technologies
7. Demers A, Keshav S, Shenker S (1989) Analysis and simulation of a fair queueing algorithm. ACM SIGCOMM Comput Commun Rev 19(4):1–12
8. Greenberg AG, Madras N (1992) How fair is fair queuing. J ACM (JACM) 39(3):568–598
9. Hiperion repository homepage. https://gitraap.i3a.info/fandujar/hiperion. Accessed 23 Oct 2020

10. InfiniBand Trade Association, et al (2020) InfiniBand architecture specification release 1.4. http://www.infinibandta.org
11. Keyes DE (2011) Exaflop/s: the why and the how. Compt Rend Mécanique 339(2–3):70–77
12. Martínez R, Alfaro FJ, Sánchez JL (2006) Decoupling the bandwidth and latency bounding for table-based schedulers. In: Proceedings of the 2006 International Conference on Parallel Processing (ICPP'06), pp 155–163. IEEE
13. Martínez R, Alfaro FJ, Sánchez JL (2009) Providing QoS with the deficit table scheduler. IEEE Trans Paral Distrib Syst 21(3):327–341
14. OpenSM Mellanox homepage. https://bit.ly/2ZC8EKD. Accessed 21 Aug 2020
15. Perftest Package homepage. https://community.mellanox.com/s/article/perftest-package. Accessed 21 Aug 2020
16. Pfister GF (2001) An introduction to the InfiniBand architecture. High Perform Mass Storage Paral I/O 42:617–632
17. RDMA aware networks programming user manual. https://bit.ly/2FDwvlX
18. Savoie L (2019) Inter-job optimization in high performance computing
19. Savoie L, Lowenthal DK, De Supinski BR, Mohror K, Jain N (2019) Mitigating inter-job interference via process-level quality-of-service. In: Proceedings of the 2019 IEEE International Conference on Cluster Computing (CLUSTER), pp 1–5. IEEE
20. Seifert R (1998) Gigabit ethernet: technology and applications for high speed LANs. Addison-Wesley Reading, Massachusetts
21. Sivaraman V (2000) End-to-end delay service in high-speed packet networks using earliest deadline first scheduling. University of California, Los Angeles
22. Souza A, Pelckmans K, Tordsson J (2020) A HPC Co-Scheduler with Reinforcement Learning
23. TOP500 homepage. https://www.top500.org. Accessed 20 Jan 2021
24. Valiant LG (1982) A scheme for fast parallel communication. SIAM J Comput 11(2):350–361
25. Yébenes P, Escudero-Sahuquillo J, Requena CG, García PJ, Alfaro FJ, Quiles FJ, Duato J (2014) Combining HoL-blocking avoidance and differentiated services in high-speed interconnects. In: Proceedings of the 21st International Conference on High Performance Computing, HiPC 2014, Goa, India, December 17–20, 2014, pp 1–10. IEEE Computer Society