On the road to build a 40 Tb/s DAQ system for the LHCb detector

Sébastien Valat[1], Matteo Manzali[2] Niko Neufeld[1], Jonathan Machen[3], Flavio Pisani[1], and Balazs Voneki[1]

[1] CERN, Meyrin Swizerland
[2] INFN, Bologna, Italy
[3] INTEL, Zurich, Switzerland

Abstract. In 2020 the LHCb detector will be upgraded during the long shutdown of the LHC. This upgrade will be used to build a hardware trigger free data acquisition system, meaning fully reading in real time all the data produced by the detector, a premier for such a big detector. The amount of data to transport will be, in total, close to 40 Tb/s requiring a special design of the network. The data fragments coming from the detectors will arrive in 500 server and will then be aggregated into a distributed way by the same 500 nodes leading to a continuous all-to-all communication pattern. Due to the amount of data and scale, this operation must operation at up to 80 Gb/s in full duplex per node to successfully handle the input flow. In order to build this system, we are now testing candidate technologies by considering fabrics coming from the HPC field and providing 100 Gb/s line rate : Intel Omni-Path and Mellanox InfiniBand. Following the theory we should be able to run this communication pattern on a full fat tree as available is most current supercomputers base on those technologies. In this paper we will show the issues we encountered while running our communication pattern on real systems. During those scaling tests, we mostly encountered issues with the real cabling and the routing tables of the machine which impact our benchmark more than expected.

Keywords: fabric,network,hpc,infiniband,omni-path,verbs,psm,mpi,fat tree

1 Introduction

Big detectors like the one used in the LHC (Large Hadron Collider) at CERN often produce a large amount of data (O(Tbits/s)) which cannot be fully read by current Data Acquisition Systems (DAQ). Hence, all the current systems rely on hardware filtering [7] to select possibly interesting collisions and reject the ones containing already well-known physics. For LHCb, it reduces the collision rate from 40 MHz to around 1 MHz [15]. Those filters are currently implemented in hardware with FPGAs. It makes them hard to maintain, operate and debug. The selected algorithms are also limited in complexity to be portable on available FPGAs. Last, those hardware triggers must be implemented with a local view

of the collision due to the impossibility to fetch all the data from the detector in one place at this stage. The hardware implementation and the local view induce physics biases which are hard to predict by simulation and to which we must accommodate. Finally, the DAQ needs to buffer the collisions until the hardware trigger take a decision (a few microseconds). This requires special electronics sometime with analog buffers. Again this adds to the complexity of the whole readout system.

In 2020 the LHCb detector will be upgraded. Since the first operation of the detector in 2008 the network technologies have made huge progress. This is mainly true for the available bandwidth when looking on HPC fabrics leading to the 100 Gb/s technologies and next coming 200 Gb/s. On this topic we can consider Mellanox InfiniBand [11], Intel Omni-Path [3] and 100 Gb/s ethernet [9] now provided by multiple companies. This is a huge gap compared to the 1 Gb/s networks currently in use by the LHCb detector. Compute performance also progressed during this period opening the chance to make a full software trigger.

Last, in the current system [2] the data fragments are sent to the aggregation nodes in burst due to the synchronization of all the readout system. In order to work in this context, the current system uses switches with deep buffers which are hard to make running and increase costs. But today, the memory of commodity server is also becoming cheap to build large buffers of multiple gigabytes avoiding to forward the buffering into costly switches.

Thanks to these evolutions, the LHCb team decided for the 2020 upgrade to build an hardware trigger free DAQ [1]. It means fully reading all the detector before making the filtering in full software in commodity servers with the complete view of the collision. This should improve the event selection efficiency from the physics view point and ease the maintainability of the system. After upgrading the sub-detectors the LHCb experiment will also produce more data with around 40 Tb/s (zero suppressed) which must be absorbed by the DAQ.

ALICE detector will also follow a similar strategy in 2020 with a little bit less data (8 Tb/s) [5]. The two other big detectors ATLAS and CMS will look forward to this approach for their next upgrade in 2025. This make us in the front line to explore this innovative approach.

In this paper we will discuss our progress on evaluating network technologies to build this DAQ considering tests done on existing HPC sites. In the first section, we will define the network topology and benchmarking communication pattern we consider. In this section we will describe our prototype application (DAQPIPE : DAQ Protocol Independent Performance Evaluator). We will follow by providing the performance results we obtained on existing HPC sites. We will then give an analysis of the impact of the network routing strategy on our applications.

2 Communication Patterns

In this section we will describe our problem giving the communication pattern we need to implement. We will also describe the benchmarks we use to obtain the results listed in this section. Finally we will discuss the network topology and technologies we consider.

2.1 New data Flow

In the new design, the data flow will start on the detector side with 10 000 optical fibers going from the underground (100 meters, where is the detector) to the surface where we will host the DAQ servers. Those links (versatile links) are protected against radiation to operate close and inside the detector for the sender side and can transfer up to 4.8 Gb/s [14]. The links will enter in a custom acquisition cards [8] developed in collaboration by CERN and French laboratories: CPPM and LAPP. This data acquisition card is able to read up to 48 optical fibers and transfer the data into the server memory via a PCI-Express generation 3 port. In total, this card is capable of reading up to 110 Gb/s to saturate the PCI-Express. This way, each server will get a fragment of the global collision and will buffer it in wait to be sent via the network.

From the logical point of view, the unit responsible of the data reading will be called a readout unit (RU). After the reception, all the fragments of a collision need to be sent via a network to be aggregated into one server. We name this stage event building. Again from the logical point, this server will be named a builder unit (BU). From a MPI perspective, the communication pattern at this stage is a kind of gather operations.

To sustain the input bandwidth, the incoming collisions must be dispatched into several builder units. Considering a network being able to sustain 80 Gb/s the system will require usage of 500 readout unit and builder unit to absorb the 40 Tb/s. In order to save money, the readout unit and builder unit will be hosted on the same nodes. This requires the ability to sustain 80 Gb/s in full duplex as each node will send and receive data to/from the network. From an MPI perspective, the global communication pattern is a all to all operation. This article will focus on this phase which we call event building.

Once the data has been aggregated into a builder unit, it needs to be sent again via networks (via a second network card) to a filter unit (FU) to be processed by the filter software selecting interesting collisions. Once reduced the data can be stored for long term at CERN for the physics analysis. The filter units need to reduce the total throughput from 40 Tb/s to around 2 Gb/s. Each builder unit will be linked to a couple of filter unit (around 8) meaning that the communication pattern for this stage will be one too many operation. We consider that we will need 4000 filter units in total. This part of the data flow is outside of the scope of this article. It will certainly be based on ethernet to save costs on the filter unit side by using the motherboard embedded network ports. Effectively, the BU need to send at 80 Gb/s, but, due to the dispatch, the filter unit receive at only 10 Gb/s.

The aggregation communication pattern contains a many to one communication patterns hence we need some buffering not to saturate the receiving node. As a server memory is now cheap, we will use the readout unit to perform buffering instead of considering using costly switches with big buffers and will control the network by scheduling the messages we send to it limiting the overflood of the network between 500 nodes. This fit well with the available InfiniBand and Omni-Path switch which not to provide deep buffers.

Finally, one can look on figure 1 to get the total data flow from the detector to the long-term storage.

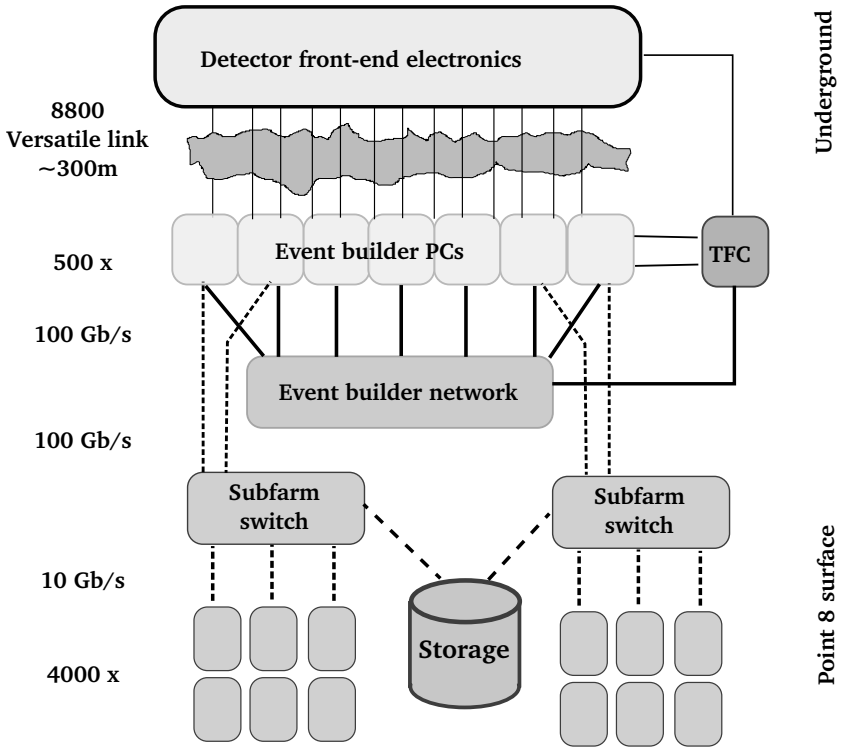

Fig. 1. Full data flow from the detector to the filter units for LHCb after the upgrade.

2.2 Network Technologies And Topology

As said in the previous section, on the networking side we can consider the requirement of 80 Gb/s for each node. Hence, we can look on the available 100 Gb/s fabrics coming from the HPC field. We can consider usage of Mellanox Infiniband, Intel Omni-Path technologies and new coming 100 Gb/s Ethernet. The first two technologies rely on a kernel bypass protocol avoiding memory copies

hence limiting the memory bandwidth and CPU consumption. This is partly done by providing an RDMA (Remote Direct Memory Operation) semantic permitting a node to directly write into a specific address on the destination so inducing zero intermediate memory copies. Ethernet now also starts to support this semantic with the RoCE [16] (Rdma over Converged Ethernet) protocol offering a nice way to save CPU cycles. On this aspect performance tests show that using TCP at 100 Gb/s requires a lot of CPU usage due to the intermediate memory copies as seen in [17] showing it requires 8 cores to feed the network at full speed with a simple point to point benchmarks.

We need to perform a kind of all to all operation so we need to provide a full bisection bandwidth between all the nodes. It leads us to consider usage of a full fat tree[10] as this topology offer the required non-blocking bandwidth between all nodes. Supercomputers equipped with InfiniBand or Omni-Path fabrics are mostly structured this way offering us a chance to test our software on existing clusters. For Infiniband and Omni-path we dispose of switches offering respectively 36 and 48 ports. This permits to fit our requirement of a full fat tree between 500 nodes within two switch levels. We can also directly consider usage of the director switches they provide with respectively 648 and 768 ports.

Both technologies will also soon provide 200 Gb/s versions but are yet not available at the time we wrote this article. Apart the higher CPU usage, the main issue to consider 100 Gb/s Ethernet is the availability of a research test site where we can test the solution at scale. This make Ethernet out of the scope of this article and forward the work depending on future opportunities.

From a prospective work we can also notice that InfiniBand is already in use in production in the DAQ of the CMS detector [6].

2.3 Our Main Benchmark: DAQPIPE

One can think using directly the all-to-all implementation of MPI but as we will show in section 3.1 it didn't scale on the tested cluster. Also, in the final production system we need to run continuously and need to handle failure recovery to continue to run even if we lose one node. This cannot be supported by using MPI. Effectively, about MPI we can find research work on failure mitigation [4] but there is no usable work about recovery.

In order to evaluate the available technologies, we built DAQPIPE [13] (DAQ Protocole Independent Performance Evaluator). This benchmark provides various communication units listed in the section 2.1 (readout unit, builder unit, filter unit) and support various network drivers, mainly:

MPI: for easier portability when performing test on HPC sites.
Verbs: InfiniBand low level protocol.
PSM2: Intel Omni-Path low level protocol.
Libfabric: Intermediate library wrapping Verbs, PSM2 protocols.
TCP/UDP: for Ethernet testing.

MPI didn't really well support threads. Hence, DAQPIPE is implemented in a totally asynchronous way to schedule the communications and ensure to

have multiple communications in flight at the same time to get more chance to saturate the network. Effectively, on InfiniBand and Omni-Path we observed that scheduling the communications one by one do not permit to reach the expected bandwidth with at best 30 Gb/s.

From the logical point of view, DAQPIPE add an event manager which is a unit assigning the events (event id) to a given builder unit. This event manager works on a credit base. Each builder unit can handle a certain amount a credit meaning that each builder unit can fetch several (credits) events at the same time. The credits are used to produce more messages to saturate the network.

DAQPIPE provide multiple protocol implementations in the way we exchange between the builder unit and readout unit. It mainly provides PUSH, PULL and PULL without event manager. In this article we will focus on the PULL protocol. In this approach the event manager assign a credit to a builder unit. Then this builder unit will send pull requests to the readout units one by one to fetch the event. We showed by using the PULL without event manager that the event manager is not a scalability issue up to the 512 targeted nodes. To order the readout unit requests DAQPIPE permit to implement various communication scheduling policies. We currently considered:

Barrel shifting: send to nodes in order starting from current node id.
Random: randomize the node communication ordering.

Remark that in the PULL protocol, the builder units control the scheduling of the communications but there is no synchronization between them. This way we avoid scalability issues due to synchronization but lose a little bit of control on the network. This also make our life easier to support failure recovery.

From the transfer point of view we know that networks like HPC fabrics often have an ideal transfer size to fully exploit the available bandwidth. On InfiniBand and Omni-Path this size is around 512 KB or 1 MB as it can be seen by using the OSU benchmark as shown in [17]. Looking at the detector shows that each collision will represent only around 100 KB in total. This amount of data is dispatched over 500 readout units meaning each unit gets only around 20 bytes per collision which is far too small to saturate the network. To solve this issue, DAQPIPE aggregate many collisions in what we call an event. Each readout unit will have a fragment of the event. To be efficient, a fragment should be around 1 MB or more, meaning that a complete event is around 500 MB. As events need to be buffered while we fully fetch them, it leads to consider using multi-gigabyte buffers on the readout unit side to have some margins in case of jitters on the network. This is where we can use the cheap memory today availability.

In addition to the data messages DAQPIPE will use command messages to synchronize the nodes. Those messages have a size of 64 bytes. Finally, to describe the collision fragments size in the data segment we will have to manage metadata segments fitting in 10 KB segments. If looking for technical details, the acquisition board will in reality produce two stream at 50 Gb/s. Hence for every data exchange we will have to send a command (request from builder unit), 2 data segments (0.5 MB) and 2 metadata segments (10 KB).

One last remark about our communication pattern. In the final system, all the readout units will produce a different data rate. Depending on the sub-detectors, some nodes will produce 40 Gb/s, 70 Gb/s or 80 Gb/s. Also the data rate will vary a little bit depending on the collisions. Those imbalances are not taken into account in this paper where we tried to reach maximal bandwidth we can achieve. But, notice it motivates the choice to use an event manager to balance the events over the builder units instead of using a static scheduling. This also seems to us more robust face to network jitters and permit to easily accommodate failure recovery.

2.4 All-to-all pattern theory on fat tree

Of course one can find in the literature some work about all-to-all communication pattern on fat trees. For this, we can refer to the work from IBM [12] about all-to-all operations on fat trees. In this document the authors provide a communication scheduling pattern to run a all-to-all pattern with half the bisection bandwidth in the spine switches. It reminds the two main algorithms to implement this operation : linear (our barrel shifting) or xor. We will show later that we observe in practice some of the theoretical pattern they provide in their paper but also observe some non describe issues with this kind of communication pattern when considering real clusters.

When we want to implement a fat tree with InfiniBand or Omni-Path we need to build each switch levels with a standard switch model hence the inter-switch links are not bigger at every level but we use several links to provide the required bandwidth. We call this kind of network clos [**?**]. The difference with the fat-tree theory comes from the routing protocol which had to choose a link to reach the destination.

On networks like InfiniBand and Omni-Path the routing protocol is base on routing tables on each switch assigning a link to reach a given destination. We will see in section **??** that this feature is the source of major limitations for our all-to-all communication pattern.

3 Benchmarking Results

In this section we will give the results we obtained over several benchmarking campaigns on existing supercomputers we granted access thanks to some companies. Even if we encountered some issues we want to give a very big thanks to those companies for offering access to their system and permit us to learn a lot about our communication pattern and our requirement on the network setup. To be fair with all of them we will not disclose the exact technology names and cluster names to only focus on the knowledge we extracted from those tests. The two tested technologies in use will be named technology A and B. For our scaling test we granted accesses to four cluster listed in figure 1. More details will be given about those clusters in the next parts of this article. Notice that only cluster A was a production supercomputer, the others were tested clusters

offered by server companies which explain the imperfection we discuss latter in this article.

Table 1. Information about clustered used in this article.

Cluster name	Technology	Tested nodes	Job Manager	Remark
A	B	512	PBS	Only half uplinks
B	B	128	Slurm	Director switch
C	A	64	Slurm	Full fat tree with ToR switches
D	A	72	PBS	Full fat tree with ToR switches

3.1 All-to-all benchmark

Before discussing DAQPIPE we can provide results with a benchmark using directly the MPI_Alltoall function called in loops and measuring bandwidth. The results obtained on cluster B and C are shown in figure 2. We will give more hints about those results in the next parts after discussing our topology analysis. Anyway this comfort us in our idea to build our own benchmark with more asynchonicity that what append in the MPI implementation which doesn't scale beyond 32 nodes (when it goes out of the leaf switches) in any of the two tested cluster. Remark that we tested IntelMPI and OpenMPI and obtained very similar results for this test showing it does not depend too much on the specific implementation. We can also notice that one of the technology (A) perform better on this operation at lower scale while the second one (B) already shows a loss of performance (10 Gb/s) at lower scale.

3.2 DAQPIPE Performance Stability

Due to its design the performance of DAQPIPE rely on several parameters which need to be tuned to extract the best bandwidth. The given parameters are:

Credits: the number of events transferred in parallel to each builder unit. Notice in DAQPIPE there is no synchronization between the event sent in parallel.

Parallel: for each credits, how many fragment requests we send at the same time. In other word, with how many nodes the builder unit communicates for each credit.

Size: the data size to fit with what works best for the network.

PPN: The number of process per node. 1 or 2 is the canonical values and 2 appeared to be the best for performance. With 2 processes per node, one process implement the RU (sending) and the other one the BU (reciving).

To summarize, the first two parameters control how much communication we have in flight at the same time to overlap the network latency and try to extract the full bandwidth. Ideally we should use 1 credit and 1 for parallel

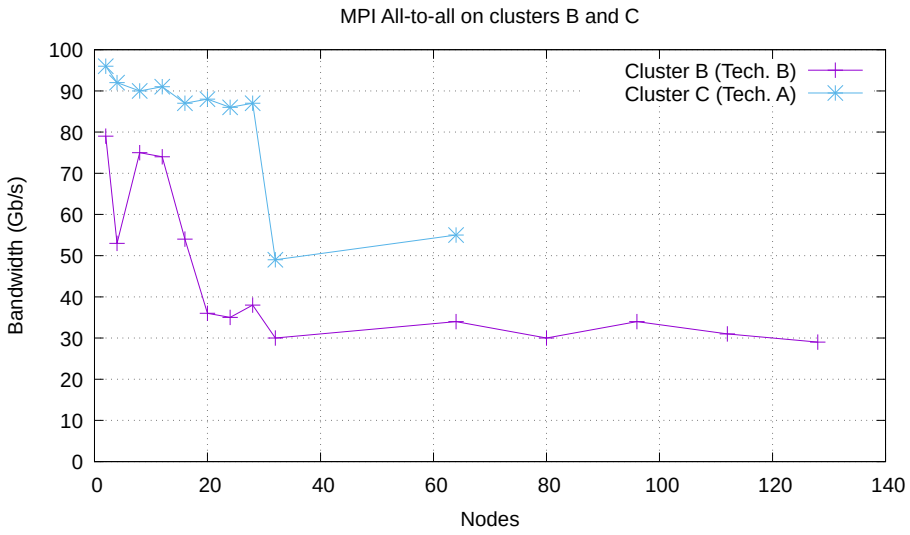

Fig. 2. Results obtained by using MPI_Alltoall function on cluster B and C.

to control exactly the scheduling of the communications. But, in practice, this approach provides bad performance with less than half the expected bandwidth. One can look on the table 2 to get the best values of those parameters on clusters of 64 nodes. We notice that we can find some magic setup working well on the two fabrics so behind dependent on the application communication pattern. But notice that the two network provides a different bandwidth for those two configuration on the tested cluster as we will explain later.

Table 2. Best configuration for DAQPIPE on Omni-Path and InfiniBand while running on 64 nodes.

	Technology A (cluster C)	Technology B (cluster B)
Credits	2	2
Parllel	8	8
Size	512 KB	512 KB
PPN	2	2
Bandwidth	83 Gb/s	60 Gb/s

Remark that many combination of those parameters gives the best bandwidth we can extract but it represents in total a relatively small surface compared to the overall parameter space. On the best technology, a quarter of the tested parameter space gives a good bandwidth on 64 nodes. Remark we will always use this parameter scan strategy to obtain the results we show in the next parts of this article.

3.3 Our First Test at Scale: 512 Nodes

In 2016 we granted exclusive access for two days on a large production supercomputer from the Top500 list for a scaling test (cluster A, technology B). Notice it was just in its first release stage at this time and was not yet in production. It has shown after our test some issues with some cables. On our side we also just finished to implement our software and didn't yet make scaling tests at mid scale before granting this access. At this stage our tests were done on 8 nodes, so, to be honest, we were not totally ready for this scale.

The cluster had a fat tree with a 2:1 ratio on the leaf switch meaning 2 downlinks (nodes) for 1 uplink to spine switches. Our application requires a full-fat tree, so, we selected thanks to the local team half the nodes on each leaf switches.

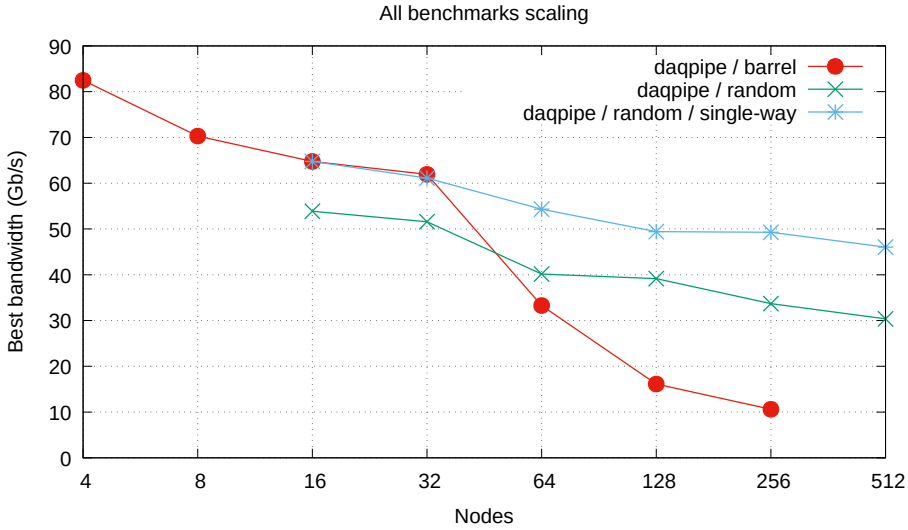

Fig. 3. Results obtained on production cluster A with various configuration. This was our first test at real scale (512 nodes) showing 17 Tb/s transported at this scale in full duplex.

The results are shown in figure 3. We firstly ran DAQPIPE using the barrel shifting mode thinking it will lead to best performance by controlling the communication scheduling. But as shown on the chart it gave us really poor performance beyond 32 nodes with 10 Gb/s on 256 nodes. This can be improved by using the random scheduling mode of DAQPIPE with 30 Gb/s on 512 nodes. We will give more explanation about this low performance in the rest of the paper. Anyway, this was already a good performance for a first-scale test as we transported in total 17 Tb/s. We remember we need to reach 40 Tb/s at this scale so we miss a factor 2.3.

One can remark the last curve of single-way mode. In this mode, each node is just sending or receiving instead of the full-duplex mode of the other lines. This shows some improvement of the performance. We will discuss again this in the latter part when looking on the routing strategy of this technology (at least for this cluster we tested). At that time we were not aware enough of the routing table impact on our application as we will discuss later.

3.4 Some Progress On Two Smaller Cluster

Thanks to a company we granted access to two smaller clusters (cluster B, 128 nodes and C 64 nodes) equipped with two technologies, respectively B and A but this time for a longer time (one week). Those two cluster where better for our test as they both provide a full fat tree, one of the cluster (B) using a director switch and the second one building its fat tree with ToR (Top of Rack) switches.

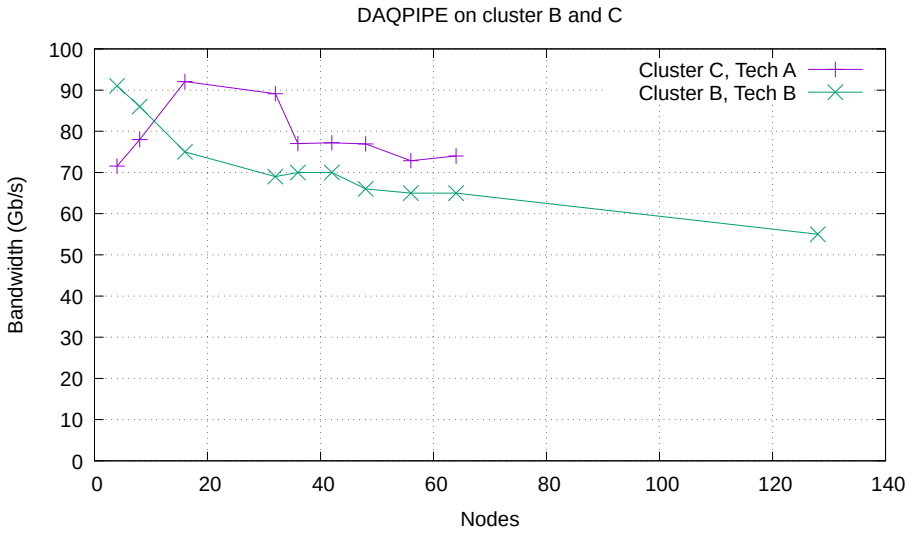

Fig. 4. Better results obtained on DELL cluster.

The results are given in figure 4. Here the technology B (the one from our previous large-scale test) show the same behavior on the director switch so we can exclude an issue with the node selection during our previous test. The technology A seems to perform better (10 Gb/s more) but again have a performance drop while scaling.

One can remark that in the two case our benchmark perform better than the MPI_Alltoall implementation which is a good point in our favor.

3.5 Micro-benchmarks

As we observed some performance issues in our first tests, we also built a series of micro-benchmarks reproduction some behaviors of DAQPIPE to see how those pattern perform on the tested network. It contains:

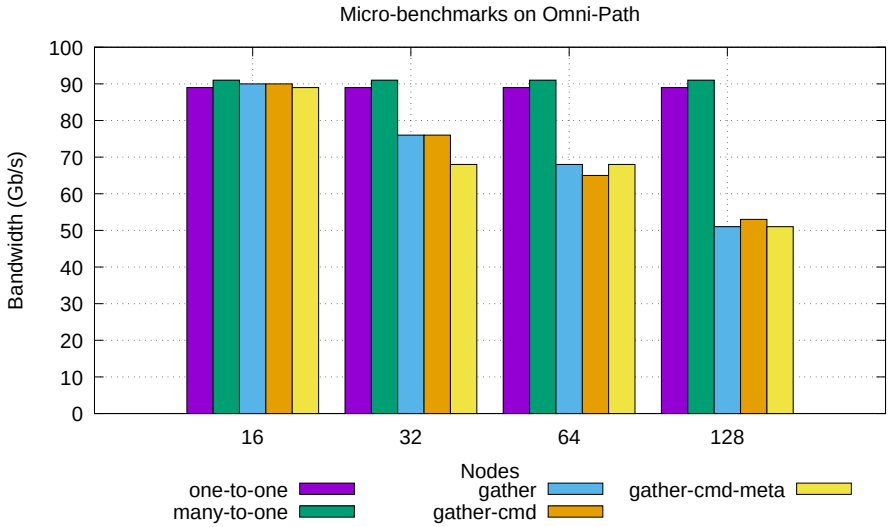

Fig. 5. Better results obtained on DELL cluster.

One-to-one: Every node sends to its neighbor. It permits to test the basic scaling of the switch. We also made a variant of this benchmark by adding a shift parameter to send to the Nth next node instead of the closest neighbor. As we will show later, it provides interesting information on possible routing issues.

Many-to-one: All the nodes send to node zero. This is basically to test the ability of the switch to provide full bandwidth on the receiving node and handle traffic congestion on the receiving link.

Gather: Reproduce the pattern of DAQPIPE but without any communication protocol. Meaning we just send data over the network selecting the target node in round robin as DAQPIPE. It uses 1 MB as message size.

Gather-with-cmd: This variant provides two types of message for every exchange. A data segment of 1 MB and a command segment of 64 bytes just like DAQPIPE does. It is mostly used to check how the network handles multi-size exchanges.

Gather-with-cmd-meta: Again similar benchmark but adding metadata segment of 10KB. It is mostly used to check how the network handles multi-size exchanges.

In order to try to understand what append, we tested those micro-benchmarks on two clusters (B and C) with two technologies. The results obtained on cluster B are presented in figure 5. Notice we obtained the same thing on the other technology. Here we observe a good scaling of the one-to-one and many-to-one micro-benchmarks but similarly to DAQPIPE we observe a performance drop on the gather benchmarks. We also confirm that the mix of message size on the network is not an issue in our benchmark.

3.6 One-to-one Benchmark With Shift Scan

We can remark that our barrel shifting scheduling can be simulated by using the one-to-one benchmark thanks to the shift parameter. This way we can reproduce each step of the all-to-all operation and see if some steps have lower performance. This is what is presented in figure 6. On this benchmark we draw the lower bandwidth of all nodes, the highest and the average. This shows an interesting behavior, with shift 0 we get a large performance drop due to local communications in MPI but this is outside of our real interest. With shift 1,2,3 and 44,45,46 we get a high bandwidth meaning we are running at full speed. Bug in the middle og this scan we observe a large performance drop. The lowest bandwidth drop at half the highest bandwidth meaning that some nodes are running with half the bandwidth. We also observe on the average that more nodes seem to be affected in the middle of the steps. This seems to show some collisions on links which we will describe in more depth in the next section.

Remark that the average bandwidth corresponds to the observation we made with DAQPIPE with a bandwidth at 46 nodes around 70 Gb/s.

4 Performance issue analysis

In the previous section we provided our scaling tests obtained on various clusters and technologies. In this section we will provide an interpretation of those results under the light of the cabling and routing analysis. We will show that our pattern is highly dependent on those two parameters. It barely confirms part of the analysis of the IBM paper but also show one missing issue to fully understand the all-to-all communication pattern on real fat trees where the inter switch links are not bigger in bandwidth but aggregating multiple links to reach it with routing rules to distribute the load depending on paths.

4.1 Impact Of Cabling

To progress we dumped the cabling and routing tables of the two clusters (B and C) to look on the exact topology they provide and make a post-mortem analysis. Looking on cluster C, we observed quickly that some nodes were miss-ordered on the leaf switches with some holes in the leaf switch ports (missing nodes or filesystem nodes).

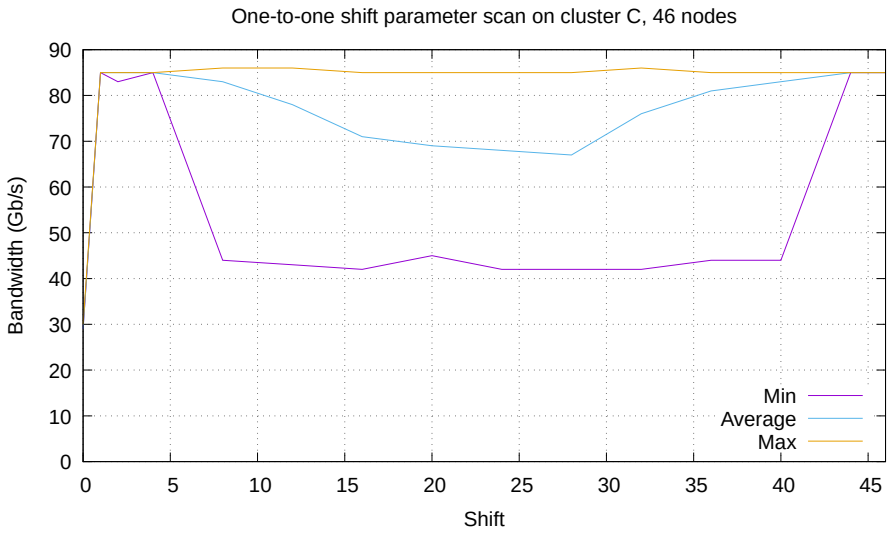

Fig. 6. One-to-one microbenchmark scan with shift parameter on cluster C to reproduce each step of the all-to-all communication pattern. It plots the maximum, average and minimum bandwidth of all the nodes for each step on 46 nodes.

To better understand we build a routing simulation in JavaScript with a display of the link conflicts. In this simulator we can inject an ideal topology (nodes well ordered and no holes) or the real topology we dumped on the cluster. We can then compute the number of conflicts on the inter-switch links (ISLs) for each step of the barrel shifting scheduling. The conflict curve match with our observation on one-to-one benchmark from section 7. We have low number of conflicts on borders and high number of conflicts in the middle.

Those simulations clearly show conflicts in the ISLs due to the randomized order of the nodes so we tried to reorder the ranks we assign to each node. This lead to the second curve showing a small improvement. But notice that the missing nodes still generate a lot of conflicts in the ISLs.

Finally we simulated the perfect cabling and routing. Then we observe a drop in conflicts but still have some. We can show by playing with the simulator that we can only have zero conflicts when we exactly fully populate the leaf switches. Playing by hand with the topology shows that the modulo of the barrel shiting create automatically link conflicts if it didn't match with the number of uplink per switch.

On the cluster B with director switch we expected a better layout. But looking on the cabling it appeared that the nodes were cabled in a "random" fashion with some sub-switches having only 4 or 8 nodes when some others have more. Also the nodes were not placed in order. Notice that in this cluster we used slurm to launch the job. As expected slurm selects the nodes based on their

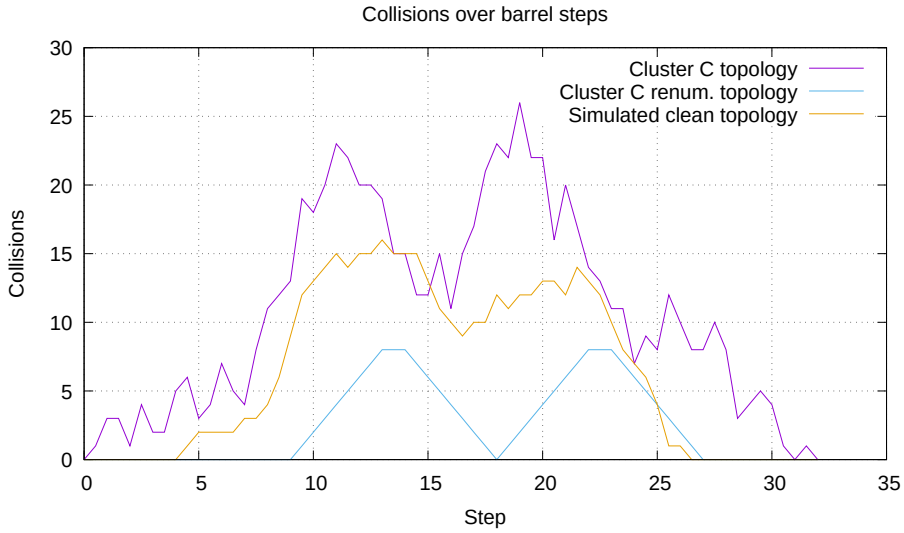

Fig. 7. Better results obtained on DELL cluster.

name (node002, node003...) and map the MPI ranks based on that order. Due to the imperfect cabling we were scheduled for a totally wrong way. This for sure explains partly why we had lower bandwidth with DAQPIPE on this cluster compared to the cluster B with other technology. But we will show in the next part that another factor was present on this technology B.

Remark that it totally explain the non-scalability of the MPI_Alltoall implementation which is heavily synchronized by making each step one by one to paying the full price on collisions making the bandwidth divided by two.

4.2 Impact Of Routing Strategies

Recently we granted again access to the big cluster A (512 nodes with technology B) for one day. This time, knowing the cabling issue we implemented a script to select the nodes we run on based on a dump of the cabling of the fabric. We also reordered the nodes base on this cabling but on this production cluster the original order of the nodes was mostly ok.

This time we started by making a scan with the one-to-one benchmark on 64 nodes hoping that the node selection will be better. We show the results into figure **??**. Her instead of getting half the bandwidth on part of the scan we observe with up to 1/3 of the bandwidth meaning we have some ISL shared between 3 streams. Again the average bandwidth matches our observation with DAQPIPE with a bandwidth of 60 Gb/s on 64 nodes.

To understand what append, we displayed the routing rules on our topology renderer and look on it. It appeared that going from first node to first node of

second switch will use a link let say 4. Then to go to first node of third switch we now use another link let say 10. We can show by simulation that not having a regular routing pattern over leaf switches leads to more conflicts due to the routing tables.

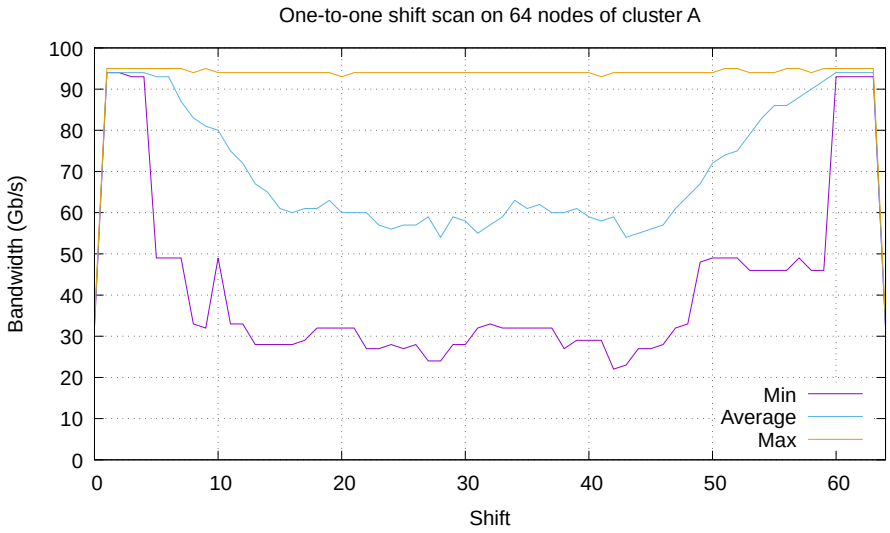

Fig. 8. One-to-one shit parameter scan on cluster A with technology B. Here is shows larger issue due to the routing tables with up to 3 conflicts per link.

4.3 A test on a cleaner topology and routing

We finally granted access to a last cluster (D) using technology B with 72 nodes. This time, the cabling of the machine was pretty perfect (still a couple of nodes miss places). Looking at the routing tables, we observed a regular pattern over all the switches. In other work, going to the Nth link of a remote switch will all the time use the Nth uplink. Thanks to this we obtained with DAQPIPE the results presented in figure 9. Here we now succeed in sustaining 83 Gb/s on 64 nodes and have a slight drop while scaling certainly due to the non-perfect cabling.

5 Conclusion

In this paper we described our progress while scaling a all-to-all communication pattern on test clusters. We show that the default MPI_Alltoall didn't scale well on two of the tested cluster. In order to get a better chance to scale and

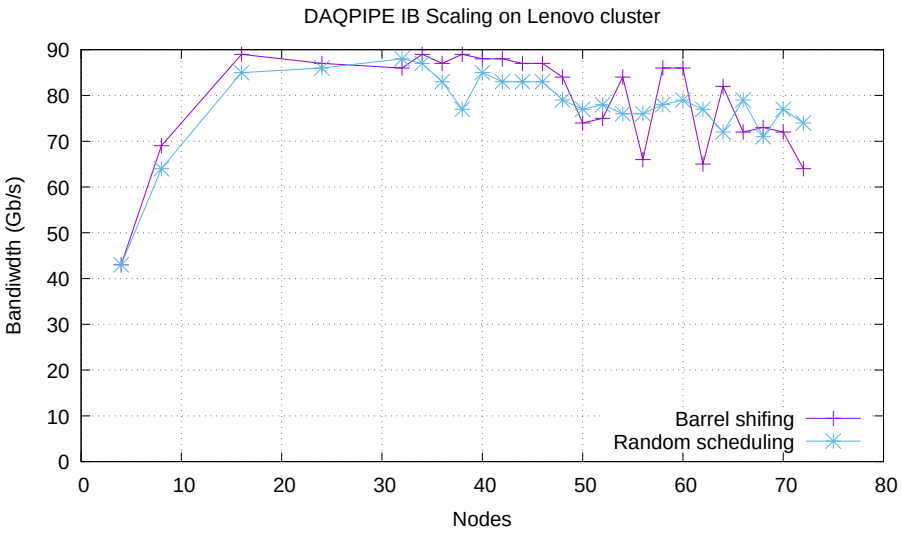

Fig. 9. Better results obtained on DELL cluster.

get support of failure recovery, we implemented our own application DAQPIPE and tried to scale if on various cluster with two technologies : InfiniBand and Omni-Path.

We observe a scalability issue with our application on most of the tested cluster and investigate the issue. This way we showed that to scale on a clos our application requires a "perfect" cabling by fully populating the leaf switches. We also show that it required an adequate routing tables to efficiently use the inter-switch links.

By testing the two technologies we show that one of the two technology (A) offer the perfect routing we need by using its fat-tree configuration. For this it produces a regular routing patter between all the leaf switch in its ISLs assignment. We observer that the other one (B) didn't provide this behavior by building a different pattern for each leaf switches. We show by simulation and real experiment that it prevents our application to scale.

Finally, we remark that on the non-perfect clusters our application perform better than the MPI_Alltoall implementation which is due to the multiple message in flight at the same time. We can maybe transpose this to provide a more robust implementation of MPI_Alltoall as in real supercomputer we for sure do not get a nice placement on the leaf switches due to the job manager so being exactly impacted by the problems we described here. We will explore this while getting again access to the tested supercomputer.

Special thanks We want to give a special thanks to INFN, Dell and Lenovo to offer exclusive accesses to their cluster for performing our performance test. We also want to thank Intel for their great help on Omni-Path.

References

1. LHCb Trigger and Online Upgrade Technical Design Report. Tech. Rep. CERN-LHCC-2014-016. LHCB-TDR-016 (May 2014), https://cds.cern.ch/record/1701361
2. Antunes-Nobrega, R., França-Barbosa: LHCb trigger system: Technical Design Report. Technical Design Report LHCb, CERN, Geneva (2003), https://cds.cern.ch/record/630828, revised version number 1 submitted on 2003-09-24 12:12:22
3. Birrittella, M.S., Debbage, M., Huggahalli, R., Kunz, J., Lovett, T., Rimmer, T., Underwood, K.D., Zak, R.C.: Intel omni-path architecture: Enabling scalable, high performance fabrics. In: 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects. pp. 1–9 (Aug 2015)
4. Bland, W., Lu, H., Seo, S., Balaji, P.: Lessons learned implementing user-level failure mitigation in mpich. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. pp. 1123–1126 (May 2015)
5. Buncic, P., Krzewicki, M., Vande Vyvre, P.: Technical Design Report for the Upgrade of the Online-Offline Computing System. Tech. Rep. CERN-LHCC-2015-006. ALICE-TDR-019 (Apr 2015), https://cds.cern.ch/record/2011297
6. Cittolin, S., Rácz, A., Sphicas, P.: CMS The TriDAS Project: Technical Design Report, Volume 2: Data Acquisition and High-Level Trigger. CMS trigger and data-acquisition project. Technical Design Report CMS, CERN, Geneva (2002), https://cds.cern.ch/record/578006
7. Dufey, J.P., Frank, M., Harris, F., Harvey, J., Jost, B., Mato, P., Mueller, E.: The lhcb trigger and data acquisition system. IEEE Transactions on Nuclear Science 47(2), 86–90 (Apr 2000)
8. Durante, P., Neufeld, N., Schwemmer, R., Marconi, U., Balbi, G., Lax, I.: 100 gbps pci-express readout for the lhcb upgrade. IEEE Transactions on Nuclear Science 62(4), 1752–1757 (Aug 2015)
9. Kissel, E., Swany, M., Tierney, B., Pouyoul, E.: Efficient wide area data transfer protocols for 100 gbps networks and beyond. In: Proceedings of the Third International Workshop on Network-Aware Data Management. pp. 3:1–3:10. NDM '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2534695.2534699
10. Leiserson, C.E., Abuhamdeh, Z.S., Douglas, D.C., Feynman, C.R., Ganmukhi, M.N., Hill, J.V., Hillis, D., Kuszmaul, B.C., St. Pierre, M.A., Wells, D.S., Wong, M.C., Yang, S.W., Zak, R.: The network architecture of the connection machine cm-5 (extended abstract). In: Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures. pp. 272–285. SPAA '92, ACM, New York, NY, USA (1992), http://doi.acm.org/10.1145/140901.141883
11. Mellanox: Introducing edr 100gb/s - enabling the use of data
12. Prisacari, B., Rodriguez, G., Minkenberg, C., Hoefler, T.: Bandwidth-optimal all-to-all exchanges in fat tree networks. In: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing. pp. 139–148. ICS '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2464996.2465434
13. Pérez, D.H.C., Schwemmer, R., Neufeld, N.: Protocol-independent event building evaluator for the lhcb daq system. IEEE Transactions on Nuclear Science 62(3), 1110–1114 (June 2015)
14. Schwemmer, R., Cachemiche, J.P., Neufeld, N., Soos, C., Troska, J., Wyllie, K.: Evaluation of 400 m, 4.8 gbit/s versatile link lengths over om3 and om4

fibres for the lhcb upgrade. Journal of Instrumentation 9(03), C03030 (2014), http://stacks.iop.org/1748-0221/9/i=03/a=C03030
15. Sciascia, B.: LHCb Run 2 Trigger Performance. PoS BEAUTY2016(LHCb-PROC-2016-020. CERN-LHCb-PROC-2016-020), 029. 7 (May 2016), https://cds.cern.ch/record/2208038
16. Tierney, B., Kissel, E., Swany, M., Pouyoul, E.: Efficient data transfer protocols for big data. In: 2012 IEEE 8th International Conference on E-Science. pp. 1–9 (Oct 2012)
17. Valat, S., Vőneki, B., Neufeld, N., Machen, J., Schwemmer, R., Pérez, D.H.C.: An evaluation of 100-gb/s lan networks for the lhcb daq upgrade. IEEE Transactions on Nuclear Science 64(6), 1480–1485 (June 2017)