Minimizing Network Contention in InfiniBand Clusters with a QoS-Aware Data-Staging Framework

Raghunath Rajachandrasekar, Jai Jaswani, Hari Subramoni and Dhabaleswar K. (DK) Panda
Network-Based Computing Laboratory
The Ohio State University
{*rajachan, jaswani, subramon, panda*}*@cse.ohio-state.edu*

***Abstract*—The rapid growth of supercomputing systems, both in scale and complexity, has been accompanied by degradation in system efficiencies. The sheer abundance of resources including millions of cores, vast amounts of physical memory and high-bandwidth networks are heavily under-utilized. This happens when the resources are time-shared amongst parallel applications that are scheduled to run on a subset of compute nodes in an exclusive manner. Several space-sharing techniques that have been proposed in the literature allow parallel applications to be co-located on compute nodes and share resources with each other. Although this leads to better system efficiencies, it also causes contention for system resources. In this work, we specifically address the problem of network contention, caused due to the sharing of network resources by parallel applications and filesystems simultaneously. We leverage the Quality-of-Service (QoS) capabilities of the widely used InfiniBand interconnect to enhance our data-staging filesystem, making it QoS-aware. This is a user-level framework that is agnostic of the filesystem and MPI implementation. Using this filesystem, we demonstrate the isolation of filesystem traffic from MPI communication traffic, thereby reducing the network contention. Experimental results show that MPI point-to-point latency can be reduced by up to 320 microseconds, and the bandwidth improved by up to 674MB/s in the presence of contention with I/O traffic. Furthermore, we were able to reduce the runtime of the AWP-ODC MPI application by about 9.89% in the presence of network contention, and also reduce the time spent in communication by the NAS CG kernel by 23.46%.**

***Keywords*-Quality-of-Service; InfiniBand; Data-Staging; Space-Sharing; Network Contention and Filesystems**

I. Introduction

There has been constant debate on the changes required to achieve exaflop performance with high-end computing machines. Several researchers and practitioners are working towards the exaflop goal by optimizing their applications, software libraries and kernels to extract every last bit of performance the underlying hardware can provide by using it in a dedicated manner. Although this approach seems to be the natural choice to make, it does not fully capture the issues brought to the table with the increase in system sizes.

*This research is supported in part by U.S. Department of Energy grants #DE-FC02-06ER25755; National Science Foundation grants #CCF-0916302, #OCI-0926691 and #CCF-0937842.

Furthermore, CPUs are getting fatter with tens of processing cores on a chip, physical memory is getting larger and matching the sizes of secondary storage systems from the past, and networks are getting faster providing enormous amount of bandwidth. With such an abundance of hardware resources, and the incapability of software components to utilize these resources, the system efficiencies are steadily dropping. Hardware components can no longer afford to be exclusive to a particular instance of an application or job at a given point in time. This explosion in the availability of resources has paved way for space-sharing mechanisms wherein multiple software components share the underlying hardware resources on the same node such as physical memory, processor, storage subsystem, and networking interconnect. But these mechanisms bring with them several challenges that need to be addressed. A major challenge that several researchers are trying to look at is the problem of resource contention.

There is also a constant need for persisting application data - either as checkpoints, or as input/output datasets. Although it is a resource-hungry operation which requires CPU cycles, network bandwidth and/or memory bandwidth depending on the type of storage system being considered, it is inevitable that the applications will have to pay the cost of these operations in exchange for data persistence. However, in space-sharing environments, the data storage operations impact the performance of all the applications in the system and not just the one that initiated them. A major aspect that is often overlooked is the interaction between the storage system and the networking interconnect. Most of the widely-used parallel filesystems such as Lustre, PVFS, GPFS, etc., provide a client-server abstraction in which there are client nodes on which applications reside, and there are a couple of object storage servers that host the application data. The data movement between these two entities involves heavy use of the interconnect system, which is also used by the parallel applications running on the client nodes for inter-process communication.

Modern interconnect technologies, such as InfiniBand (IB), which have been dominating the commodity networking space in the High-End Computing (HEC) domain provide several novel features that can be used to avoid

contention and congestion. Of particular interest in this context is its Quality-of-Service (QoS) capability that provides a notion of dedicated bandwidth and controlled latency to selected traffic flows. This opens-up the possibility of alleviating network contention problems, but not completely eliminating it, by carefully orchestrating the data-flow from different components in the system using the same physical interconnect fabric. With this as our motivation, we would like to address the following open research challenges:

1) Can the performance impact of network space-sharing between parallel applications and filesystems be characterized?
2) How can the QoS capabilities provided by cutting-edge interconnect technologies be leveraged by parallel filesystems to minimize network contention?
3) How can existing HPC middleware benefit from such a QoS-enabled parallel filesystem?

II. QoS support in InfiniBand

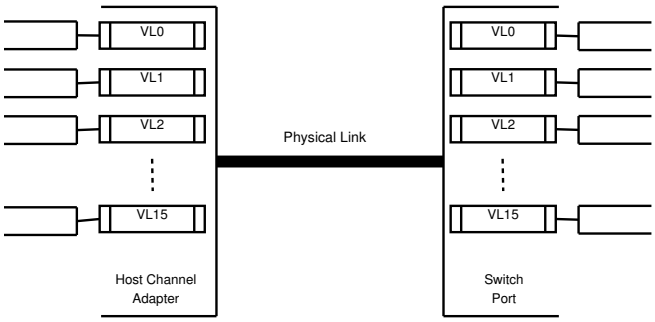


Figure 1. Per-VL Buffering and Flow control

InfiniBand is an open-standard high-speed interconnect that provides send-receive semantics, and memory-based semantics called Remote Direct Memory Access (RDMA). RDMA operations allow a node to directly access a remote nodes memory contents without using the CPU at the remote side. These operations are transparent at the remote end since they do not involve the remote CPU in the communication. InfiniBand empowers many of todays Top500 Supercomputers.

Fundamentally, network QoS mechanisms provide a way to either increase or limit the priority of data flow based on level of importance given to it when configuring a network. The InfiniBand interconnect architecture provides support for QoS using abstractions called *Service Level (SL)* and *Traffic Class (TClass)* at the switch-level and router-level, respectively. These abstractions hide-away the underlying components that help achieving QoS - Virtual Lanes (VL), Virtual Lane arbitration and link-level flow control. Virtual Lanes allow multiple independent data flows to share the same physical link, but with exclusive buffering and flow control resources. Figure. 1 illustrates how the IB HCA and switch ports use the same physical link but provide independent buffers for each VL. A VL arbiter controls the link usage by selecting appropriate data flows based on a VL arbitration table. The IB specification allows implementers to provide up to 16 VLs - 0 to 15, with VL15 reserved for management traffic and VL0-14 reserved for general purpose traffic.

A Service Level (SL) is a 4-bit field in the Local Routing Header of a packet that indicates a class of service that will be offered to that packet. Each component in an IB network maintains a Service Level to Virtual Lane (SL2VL) mapping table that is consulted before sending a packet on a given SL. The SL to VL mapping and the priorities set for each VL are configurable and is deployment-specific. The TClass field in a Global Routing Header mimics the functionality of the SL field, but only at the router-level. All these parameters are configured within the InfiniBand subnet manager.

We have leveraged these capabilities of the InfiniBand interconnect system in our hierarchical data-staging architecture, making it a QoS-aware parallel filesystem. The fundamentals of the data-staging architecture are discussed in the following section.

III. Hierarchical Data Staging

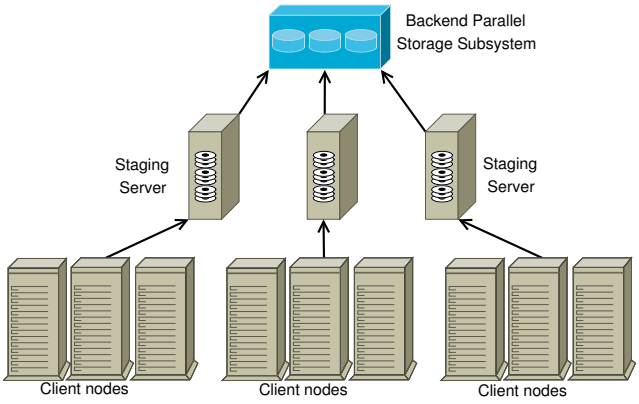


Figure 2. Hierarchical Data-Staging Framework

For the purpose of this work, we enhance the hierarchical data-staging architecture developed as a part of prior research [1] that evaluated the benefits of such a system to checkpointing mechanisms. Essentially, this staging system is a filesystem by itself, which is an interposition layer between the kernel's Virtual Filesystem Switch (VFS) and the client nodes. In a typical configuration, a group of client nodes are assigned to a staging server, the combination of which forms a staging group. All the client nodes in a cluster, together with a set of dedicated staging servers form an array of such groups. The staging servers, which are equipped with fast non-volatile storage devices like SSDs, are capable of absorbing heavy bursts of I/O from the client nodes during

operations such as checkpointing, out-of-core processing, in-situ visualization, saving a computation's output data, etc. The application running on the client nodes can proceed with its computation as soon as it hands-over the data to these intermediary nodes, instead of blocking on the write operation until the entire data is written to stable storage. These staging servers are equipped with a *Data Mover* component, which then streams the data to the backend parallel filesystem in a lazy manner, i.e, when the staging servers are not servicing any client nodes and the network is idle. This data movement from the staging servers to the backend filesystem happens over a network transport that is supported by the corresponding parallel filesystem. In modern clusters, this is typically an exclusive network for the storage sub-system's use, and I/O traffic on this network fabric would not interfere with communication traffic from user applications.

The initial data movement from the client nodes to the staging servers uses the InfiniBand RDMA protocol to copy the data from the client node's memory to that of the staging server directly, without involving the client's processor. However, this high-bandwidth data movement interferes with the communication streams from other applications that use the same network fabric. The contention between the I/O traffic and communication traffic is more prominent in the case of space-sharing applications, in which case the I/O traffic from one application introduces heavy noise which affects the communication latency and bandwidth of the other. We aim to address this problem by making the staging architecture QoS-aware.

IV. Design Goals and Alternatives

For a solution that is aiming to address the problem of network contention between I/O traffic and communication traffic from parallel applications, several design choices are available. Figure 3 illustrates the architecture of some of these choices. This discussion assumes that the parallel applications follow the Message-Passing Interface (MPI) parallel programming model [2], but the proposed designs are applicable to any parallel programming model that can take advantage of the InfiniBand interconnect. Prior work from our research group [3] studied the benefits of using multiple VLs to avoid congestion due to Head-of-Line blocking by leveraging the QoS capabilities of InfiniBand inside our MPI library - MVAPICH2 [4]. Such a technique can be used to avoid contention due to I/O by avoiding the default SL, which is SL0, over which I/O traffic is normally moved. But such a technique will be implementation-specific and will not be portable, requiring a redesign of each MPI library that wants to use it. The second potential design alternative is to modify parallel filesystem kernels (Figure 3(b)), such as Lustre [5] or PVFS2 [6], making it QoS-aware in such a fashion that it can isolate its own I/O traffic to a non-standard SL and let the communication traffic from MPI applications stream freely over the default SL. Yet again, such a solution will be specific to the filesystem that was enhanced, and can not be ported to be a system-wide solution for this problem when multiple filesystems are used for varying purposes including checkpointing, scratch-space, etc.

The approach we have chosen however, achieves both portability and performance by leveraging the QoS capabilities of the IB subsystem in our data-staging framework, as shown in Figure 3(c). This is a user-level framework that sits between the backend parallel filesystems and the MPI library making it implementation-agnostic. This framework can intercept the I/O calls that are directed to the backend storage from any of the I/O interfaces, such as POSIX-I/O, MPI-I/O, HDF5 and NetCDF, that the parallel application might be using to write data.

V. Detailed Design

A. Configuring the IB Subnet Fabric

The core-logic of the InfiniBand QoS mechanism is configured within the subnet manager. For the purpose of this work, we have used OpenSM, which is a widely used open-source implementation of the InfiniBand subnet manager specification. It is comprised of the Subnet Manager (SM) that initiates and configures the IB subnet, and the Subnet Management Agent (SMA) that is deployed on every device port to monitor their respective hosts. The SM and SMAs communicate using Subnet Management Packets (SMP), which uses the exclusive management lane - VL15 for its traffic.

As discussed in Section II, OpenSM assigns weights to different VLs based on a VL arbitration table. It also maintains an SL2VL mapping, which is correlated with the arbitration table to identify the weights for different service levels. These two key parameters, in addition to few others, are provided to OpenSM during initialization using a configuration file - `opensm.conf`. Figure 4 is a snapshot of the configuration file used for this work. The figure enlists the different parameters that are set inside this file.

The `qos_ca_max_vls` parameter specifies the maximum number of VLs any given Host Channel Adapter in the subnet can support. Most of the current generation HCAs support only up to 8 VLs. The VL arbitration table consists of two sub-tables: a high-priority and a low-priority one. The VL arbiter employs a weighted round-robin scheme and processes the high-priority table first, followed by the low-priority one, giving each VL the turn to send the corresponding amount of data.

`qos_ca_high_limit` is the maximum number of packets that can be sent by a VL in the high-priority table before yielding to those in the low-priority list. This is to ensure that the low-priority VLs do not get starved for a turn. `qos_ca_vlarb_high` and `qos_ca_vlarb_low` specify the number of 64-byte data units a given VL can send when its turn arrives. `qos_ca_sl2vl` lists the VLs

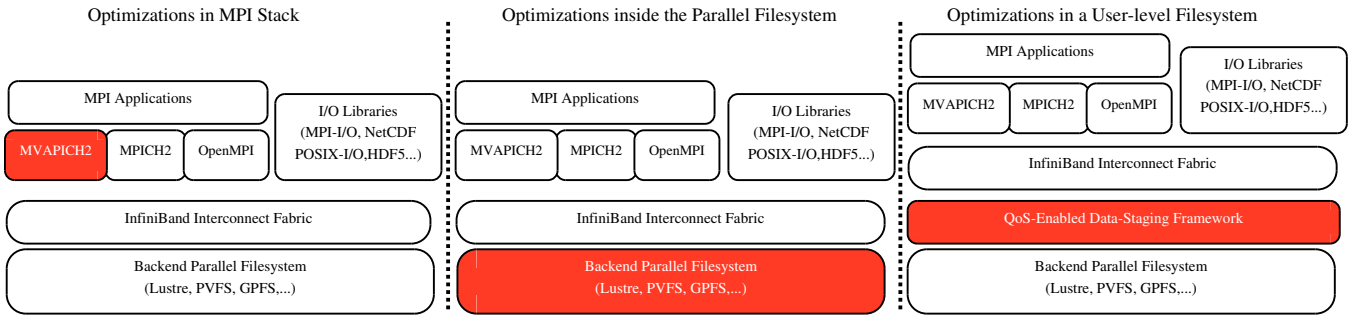


Figure 3. Design Alternatives

that should be mapped to the corresponding SLs in order. Our HCA supports only 8 general purpose VLs, so the SLs 8-15 are mapped to the management VL15. The `*_swe_*` variants of these parameters have the same significance, but for the ports on the switches.

Our arbitration table is configured such that, SL0 gets the highest priority. The weights of the other SLs are set to ”0” in the high-priority lane. This ensures that the non-zero SLs do not send data over the link as long as SL0 as a packet to send. The non-zero SLs get a turn to send data only when there are no packets in SL0’s queue, or if the `qos_ca_high_limit` has been reached. SLs 1-7, listed in the low-priority list, have been given weights in increasing powers of 2 to study the impact of SL weights.

B. Enabling Quality-of-Service in the Filesystem

InfiniBand uses a memory-based communication abstraction whose interface is known as a Queue-Pair (QP). The QP, which is also the virtual end-point of a communication link, is used to achieve direct memory-to-memory transfers between different applications and system software components. In the hierarchical data-staging architecture, the QPs are persistent, in the sense that they are created when the staging filesystem is mounted on the client nodes. There is a QP-based connection established between every client and the staging server that governs it. There is no connection established amongst the clients themselves as the data is not moved between them, but it is only flushed to the burst-buffers available with the staging server.

One of the goals of making the staging filesystem QoS-aware is to give the highest priority to the network communication traffic from the parallel applications, and to move the I/O data only during idle times when the application is busy in a compute-phase. Another design goal was to be able to set the SL over which the I/O packets from the clients are sent, at runtime when mounting the filesystem, and dynamically based on certain hints provided to the filesystem during the file write. To this effect, the filesystem was enhanced to establish an array of QPs, each of which was attributed to a different non-zero SL ranging from 1 through 7. The SL over which the file traffic from the client nodes should be sent can be set by one of two methods.

One way is to set an environment variable (`STAGEFS_SERVICE_LEVEL=<n>`) when mounting the filesystem, where `"n"` is the SL number to use. With this method, all of the filesystem traffic will use the SL-n. The default Service Level that is used is 1, which has the least weight in the low-priority portion of the VL arbitration table. SL1 was chosen as the default so that all file I/O is provided link access only in a best-effort manner, without giving an opportunity for contention that affect other applications’ communication latencies. The other method is to provide hints to the filesystem using certain filename-suffixes when writing the file at the client-side.

```
qos_ca_max_vls          8
qos_ca_high_limit       255
qos_ca_vlarb_high       0:40,1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0
qos_ca_vlarb_low        0:0,1:2,2:4,3:8,4:16,5:32,6:64,7:128
qos_ca_sl2vl            0,1,2,3,4,5,6,7,15,15,15,15,15,15,15,15
qos_swe_max_vls         8
qos_swe_high_limit      255
qos_swe_vlarb_high      0:40,1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0
qos_swe_vlarb_low       0:0,1:2,2:4,3:8,4:16,5:32,6:64,7:128
qos_swe_sl2vl           0,1,2,3,4,5,6,7,15,15,15,15,15,15,15,15
log_flags               0x0F
```

Figure 4. OpenSM Configuration File

Adding the suffix ".sl<n>" to file names will hint the filesystem, letting it know of the Service Lane that needs to be used to send this particular file to a staging server. This allows for a per-file QoS with which applications that perform heavy I/O operations can prioritize data, such as giving higher priority to in-situ processing data that is needed for the next step of the computation, and lower priority to application log files.

As indicated in Section III, the data movement from the staging servers to the backend filesystem does not contribute to network contention as it happens in a lazy manner when the network is idle, and so, it does not have to be aware of the Service Level abstraction. If the backend parallel filesystem does support InfiniBand transport, the data packets will just be sent over the default SL0.

VI. Experimental Evaluation

In this section, we present the results of our experiments that evaluate the benefits of the proposed design with some micro-benchmarks and some real-world MPI applications. The computing cluster used for these evaluations, Cluster A, is a 160-node Linux-based system. Each compute node has eight Intel Xeon cores organized as two sockets with four cores per socket and has 12 GB of memory. They are equipped with InfiniBand QDR Mellanox ConnectX-2 HCAs, which support up to 8 Virtual Lanes. The operating system used is Red Hat Enterprise Linux Server release 6 with the 2.6.32-71.el6.x86_64 kernel. Cluster A also has 16 dedicated storage nodes with the same configuration, but with 24GB of RAM each. Additionally, the storage nodes are equipped with a 300GB OCZ VeloDrive PCIe SSD.

A. Micro-Benchmark Evaluation

In these set of experiments, we study the impact of filesystem noise on MPI communication by characterizing its latency, bandwidth and some Representative MPI collective operation latency, with varying message sizes. The OSU Micro-Benchmarks (OMB) [7] suite was used for these evaluations. All tests were run with 2 MPI processes residing on two unique client nodes, with the filesystem noise generated by one I/O thread on each of these nodes that writes a stream of raw data in 1MB blocks to the staging system using the `dd` GNU core utility.

Direct I/O was used when writing the raw I/0 data to avoid kernel page cache buffering. This was done to get a more accurate representation of the data being written. The MPI processes and the I/O writers are mapped to different processing cores of host CPUs. A single storage node which houses a 300GB SSD is setup to be the staging server that governs these two client nodes. The fast-SSD storage helps alleviate the storage bottlenecks, and to get a clearer understanding of the impact on the InfiniBand network.

From Figure 5, it can be observed that the latency of MPI application communication gets hurt when the filesystem traffic shares SL0 with MPI traffic. The impact of this noise is more prominent with larger message sizes, as seen in Figure 5(c) where the contention in the network is increasing the latency by up to 400 microseconds for a message of size 4MB. However, when using the QoS-aware filesystem to isolate the I/O traffic to a non-zero SL (SL1 in this case), the latency is lesser than the SL0 case by 320 microseconds, which is merely 80 microseconds more than the default latency. A similar trend can also be observed with the bandwidth numbers in Figure 6. The I/O traffic that is flowing over SL0 degrades the bandwidth by up to 700 MB/s for a 4MB message. However, when this traffic is channeled over SL1 to segregate the I/O and MPI traffic, the bandwidth degradation is reduced by up tp 674MB/s.

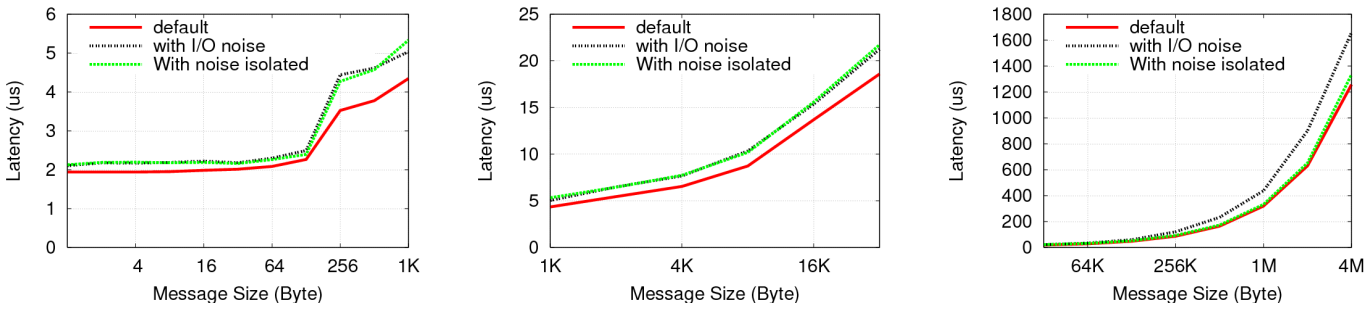
The impact of file I/O noise is no different in case of collective operations. Figure 7 shows how the operation latency of a representative MPI Collective (MPI_AlltoAll) is affected. The latency goes up by about 250 microseconds for a 1MB message. Using SL1 for I/O brings down this latency to about 15 microseconds more than the default latency.

From the above experiments, it is clear that the proposed solution can benefit both latency-sensitive and bandwidth sensitive MPI applications. The benefits of the QoS-based I/O noise isolation becomes more prominent with increasing message sizes. This is beneficial for most of the communication-intensive MPI applications which intuitively aggregates several small messages to form a large message locally before sending it out to a different MPI rank using the network, rather than paying the overhead costs of sending several smaller messages.

B. Impact of SL Weights

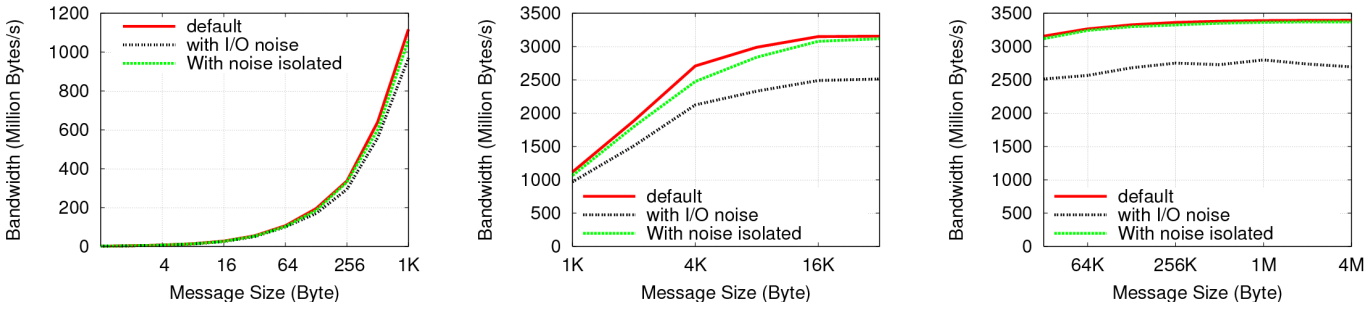
It is also desirable to understand the impact of the credit weights that were assigned to the different SLs, as described in Section V-A. We yet again use the OSU Micro-Benchmarks for this evaluation, with the same type of filesystem noise as before. Figure 8 shows the trends for the latency, bandwidth and bi-directional bandwidth of MPI Point-to-Point operations, and Figure 9 illustrates the trends for two representative collective operations: `MPI_AllReduce` and `MPI_AlltoAll`. All the graphs in this experiment illustrate the trends only for large messages, as the smaller and medium message did not have a significant impact as far as performance goes, as seen in the earlier experiments.

One thing to note is that the MPI communication traffic is *always* sent over SL0, which is also the default Service Level. The SL over which the filesystem traffic was channeled was varied between 1 and 7, and the corresponding behavior of the MPI performance was noticed. Both in the case of point-to-point and collective operations, there is an insignificant difference in performance as the SL is varied for the I/O flow. The reason for this can be attributed to the way the SLs weights were assigned (see Section. V-A). As



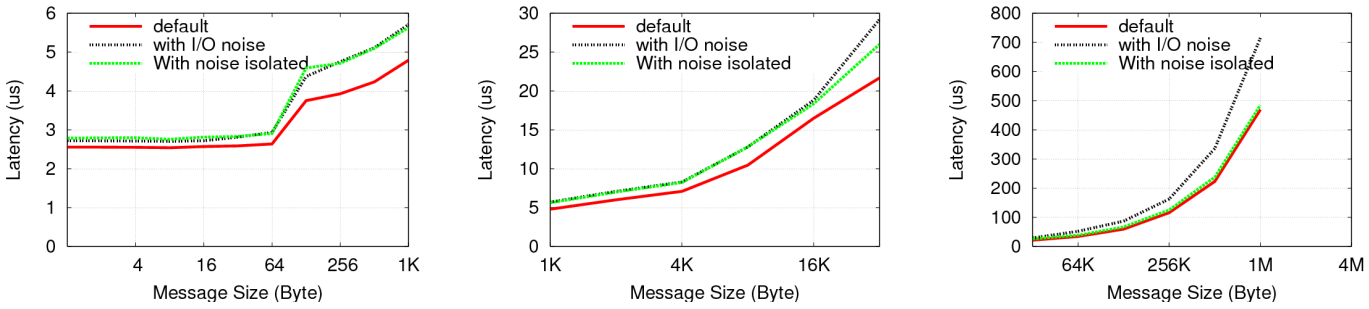
(a) Small message latency (b) Medium message latency (c) Large message latency

Figure 5. Impact of I/O noise on MPI Pt-to-Pt Latency (Lower is better)



(a) Small message bandwidth (b) Medium message bandwidth (c) Large message bandwidth

Figure 6. Impact of I/O noise on MPI Pt-to-Pt Bandwidth (Higher is better)



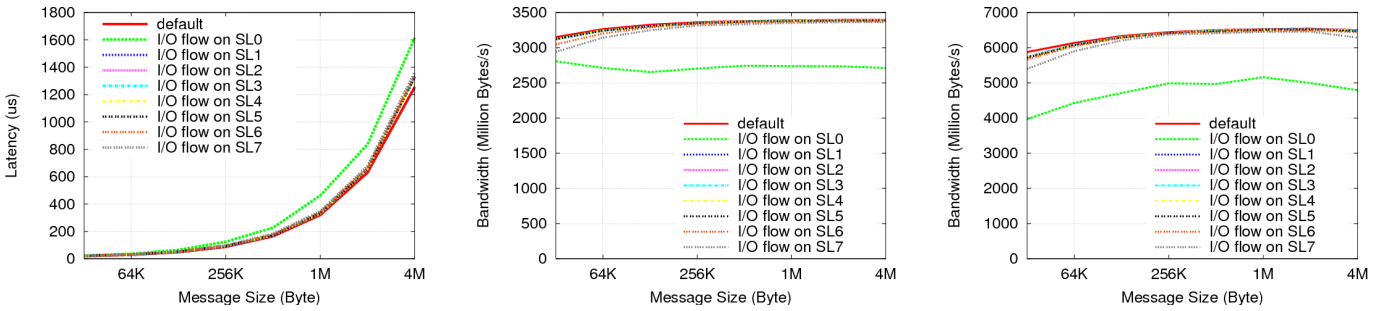
(a) Small message latency (b) Medium message latency (c) Large message bandwidth

Figure 7. Impact of I/O noise on MPI_AlltoAll Collective Latency (Lower is better)

long as SL0 has a data packet to send, SL1-7 which has an entry only in the low-priority list, will not get a chance to send data unless the `qos_ca_high_limit` is reached. Based on weights set in the `opensm.conf` file (shown in Figure 4), it is evident that the SLs 1-7 are serviced only in a best-effort manner.

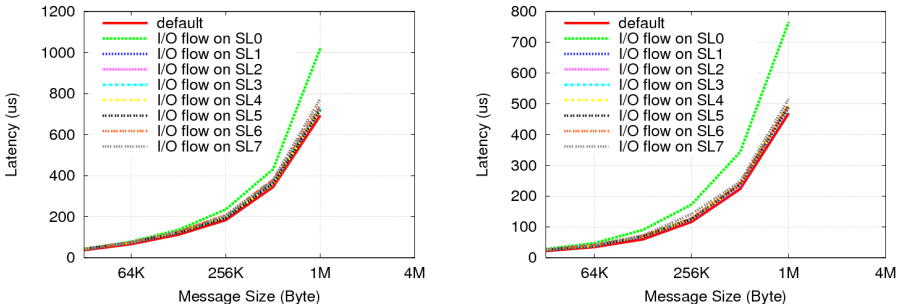
C. Impact on Applications

To study the impact of our designs on real applications, we evaluated it with the Anelastic Wave Propagation (AWP-ODC) MPI application which simulates dynamic rupture and wave propagation during an earthquake in 3D. This application has been scaled to more than hundred thousand processing cores. This application is both computation and communication intensive. In between computations, velocity values are exchanged with processes containing neighboring data sub-grids in all directions of the 3D process grid - north, south, east, west, up and down. For our runs, we set the input data-grid values NX, NY, NZ to 256. The application was run on 64 cores of cluster A, with the 3D process grid organized as a 4x4x4 matrix. TMAX value was set to 20, making the application run with 2,001 iterations. Figure 10(a) shows the normalized runtime of this



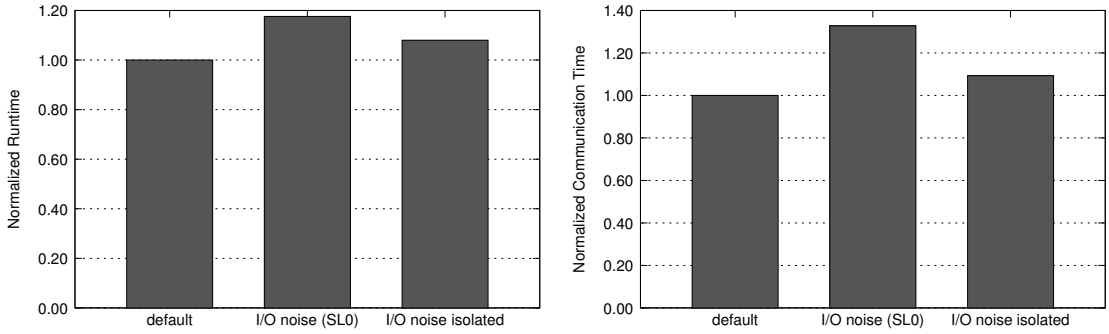
(a) Large message latency (b) Large message bandwidth (c) Large message bi-directional bandwidth

Figure 8. Impact of Service Level Credit Weights on Pt-to-Pt operations (in the presence of I/O noise)



(a) AllReduce large message latency (b) AlltoAll large message latency

Figure 9. Impact of Service Level Credit Weights on Collective operations (in the presence of I/O noise)



(a) Anelastic Wave Propagation application (b) NAS Parallel BenchmarkConjugate Gradient (CG.D.64)

application in the default case, with I/O noise on the default SL0, and with the I/O noise isolated to SL1 using the QoS-aware filesystem. The filesystem noise was generated in the same manner as in the previous experiments. Each of these runtimes have been normalized against the default runtime. The runtime with I/O traffic going over the same SL as the application increased the runtime by 17.9% owing to the contention caused at the network HCA and links. Isolating the filesystem traffic however reduces this overhead and brings down the runtime to just 8% more than the default runtime.

A similar trend can also be observed with other applications like the CG kernel from the NAS benchmark suite, which is a Conjugate Gradient method to solve unstructured sparse linear systems. The kernel runs on a 2D grid of power-of-two number of processes. There are two transpose processor communications involved that updates a vector element across iterations. Figure 10(b) shows the normalized network communication time of the CG kernel running with 64 MPI processes across 8-client nodes, with I/O noise on SL0 and with the I/O noise isolated to SL1. The time spent in communication is 32.77% more than the default time spent in the former case, but just 9.31% more in the latter.

VII. Related Work

Providing QoS from a filesystem's perspective might mean several things. Several researchers have looked at

techniques to allow applications use portions of the disk bandwidth exclusively [8], [9], [10]. This involves providing QoS from a disk-scheduling perspective. Wu and Brandt [11] add QoS support to the Ceph [12] Object-Based filesystem. This work however, focuses on the disk bandwidth and not the network performance. Likewise, Zhang et al. [13] have proposed a QoS mechanism for the PVFS2 filesystem, which is based on machine-learning techniques that can translate program execution time goals into I/O throughput bounds. Xu et al. [14] propose a virtualization based bandwidth management framework for parallel filesystems where they employ proxy servers which provide differential services to applications based on a predefined resource sharing algorithm. Our work is orthogonal to the above discussed projects that solely focus on providing differentiated services for the end-users based on the storage subsystem throughput. These solutions are not portable, as discussed in Section IV, owing to their implementation-specific designs.

Many researchers have also investigated the use of InfiniBand QoS capabilities. Prior work from our research group [3] proposed the simultaneous use of multiple virtual lanes that are provided by IB to avoid Head-of-Line congestion. This work was the first of its kind, which made use of multiple virtual lanes at the MPI level. Alfaro [15] proposed an optimal configuration for the VL arbitration table to optimize performance. Alfaro et al. [16] also proposed a formal model to manage the arbitration tables in InfiniBand. The same group worked on a framework to provide QoS over advanced switching systems [17]. All these solutions focus on the theoretical aspects of InfiniBand QoS specification, and can work in conjunction with the work presented in our paper.

Summary and Future Plans

In this work, we have developed a data-staging framework filesystem that takes advantage of the QoS features of InfiniBand network fabric to reduce the contention in the network by isolating the I/O data flow from the MPI communication flow. This is a portable solution that can work with any MPI library and any backend parallel filesystem in a pluggable manner. We have also studied the impact of our solution with representative micro-benchmarks and real applications. Experimental results show that with the proposed solution, the point-to-point latency of MPI applications in the presence of I/O traffic can be reduced by up to 320 microseconds for 4MB message size, and the bandwidth for which can be increased by up to 674MB/s. Collective operations such as MPI_AlltoAll could also benefit from this work, with its operation latency reducing by about 235 microseconds in the presence of filesystem noise. The AWP-ODC MPI application's runtime in the presence of I/O traffic was reduced by about 9.89%. The time spent in communication by the CG kernel with I/O traffic was reduced by 23.46%

As part of the future work, we plan on carrying out experiments at a larger scale. We also plan on studying the impact of this solution on workloads from multiple I/O libraries like MPI-I/O, NetCDF, HDF5, etc. These I/O libraries are predominantly used by several MPI applications to write checkpoint data and working datasets. Furthermore, we would like to evaluate the impact of such a best-effort service given to the I/O traffic on real-world filesystem workloads. Such a study would help in determining a balanced threshold which would reduce the contention for both MPI workloads and I/O workloads without compromising on the performance of one over the other.

References

[1] R. Rajachandrasekar, X. Ouyang, X. Besseron, V. Meshram, and D. K. Panda, "Can Checkpoint/Restart Mechanisms Benefit from Hierarchical Data Staging?" in *Workshop on Resiliency in High-Performance Computing*, 2011.

[2] "Message Passing Interface Forum," http://www.mpi-forum.org.

[3] H. Subramoni, P. Lai, S. Sur, and D. K. D. Panda, "Improving Application Performance and Predictability Using Multiple Virtual Lanes in Modern Multi-core InfiniBand Clusters," in *Proceedings of the 2010 39th International Conference on Parallel Processing*, ser. ICPP '10, 2010.

[4] "MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE," http://mvapich.cse.ohio-state.edu.

[5] "Lustre Parallel Filesystem," http://www.lustre.org.

[6] "PVFS2 Parallel Filesystem," http://www.pvfs.org.

[7] "OSU Micro-Benchmark Suite," http://mvapich.cse.ohio-state.edu/benchmarks.

[8] W. G. Aref, K. El-Bassyouni, I. Kamel, and M. F. Mokbel, "Scalable QoS-Aware Disk-Scheduling," in *Proceedings of the 2002 International Symposium on Database Engineering and Applications*, ser. IDEAS '02, 2002.

[9] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, and A. Silberschatz, "Disk Scheduling with Quality of Service Guarantees," in *Multimedia Computing and Systems, 1999. IEEE International Conference on*, 1999.

[10] S. Iyer and P. Druschel, "Anticipatory Scheduling: a Disk Scheduling Framework to Overcome Deceptive Idleness in Synchronous I/O," *SIGOPS Oper. Syst. Rev.*, 2001.

[11] J. C. Wu and S. A. Brandt, "Providing Quality of Service Support in Object-Based File System," in *Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*, ser. MSST '07, 2007.

[12] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System," in *Proceedings of the 7th symposium on Operating systems design and implementation*, ser. OSDI '06, 2006.

[13] X. Zhang, K. Davis, and S. Jiang, "QoS Support for End Users of I/O-Intensive Applications Using Shared Storage Systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11, 2011.

[14] Y. Xu, L. Wang, D. Arteaga, M. Zhao, Y. Liu, and R. Figueiredo, "Virtualization-based bandwidth management for parallel storage systems," in *Petascale Data Storage Workshop (PDSW), 2010 5th*, 2010.

[15] F. J. Alfaro, "A Strategy to Compute the InfiniBand Arbitration Tables," in *Proceedings of the 16th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '02, 2002.

[16] F. Alfaro, J. Sanchez, M. Menduina, and J. Duato, "A Formal Model to Manage the InfiniBand Arbitration Tables Providing QoS," *IEEE Trans. Comput.*, 2007.

[17] R. Martínez, F. J. Alfaro, and J. L. Sánchez, "A Framework to Provide Quality of Service over Advanced Switching," *IEEE Trans. Parallel Distrib. Syst.*, 2008.