Parallel I/O Profiling

Aditya Rohan (160053), Anshul Vijayvergiya (150113)

ABSTRACT

I/O is a bottleneck in almost all computing systems, owing to the fact that the I/O speeds are not at the same level as processor frequency. Increasing the number of processes accessing the same file for I/O doesn't help curb this issue. Identifying the bottlenecks in parallel systems is more important than ever in the era of supercomputers trying to predict things in real-time. In this project we try to profile the I/O performance of the HPC2010 supercomputer at IITK to identify any possible bottlenecks that can be related to the topology of the supercomputer nodes.

1. INTRODUCTION

The gap between the advances made in storage technology and processor performance has pronounced negative side-effect on the modern high performance computing subsystems. While the improvement in the technology itself is a long and lengthy process, we can readily develop and implement software-based optimized solutions. Understanding the complex parallel I/O operations is critically important to propose an I/O solution to improve performance.

Unfortunately, the parallelism in HPCs makes understanding IO in such cases extremely difficult due to multiple software and hardware layers managing IO while maintaining consistency and concurrency. Figure 1 shows the IO software stack of a parallel IO system. The storage hardware is the bottom most level which only contacts the parallel file-systems such as PVFS[2], GPFS[3], Lustre[4] and PanFS[5]. The parallel file-system manages the access of multiple processes to same files and directories. The MPI-IO library provides the IO interface for process coordination during IO. There are also multiple high level libraries that communicate with the storage through the MPI-IO library. From Figure 1 we can also see that some applications can directly communicate with the MPI-IO or POSIX-IO library. This multipath communication makes the whole IO subsystems very complex. Therefore, before we can propose any solutions to optimize IO performance we must first find the possible bottlenecks.

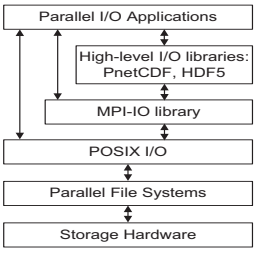

Figure 1: Parallel I/O Software Stack [1]

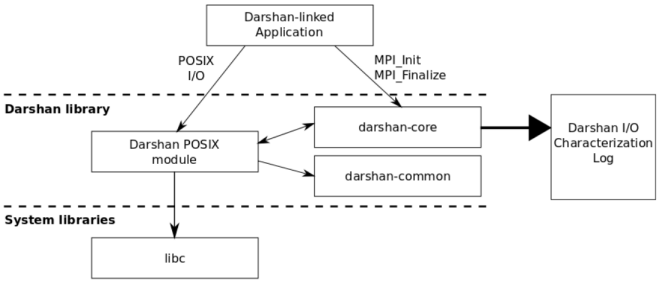

Figure 2: Darshan Runtime Environment

2. RELATED WORK

Much of the related work in this area involves development of tools for characterization, instrumentation and profiling of parallel applications. One such widely used tool is Darshan[6]. Darshan is a lightweight toolkit for characterizing IO performance in HPC applications. Darshan has two primary components which can be used for instrumentation,

- **darshan-runtime**: instruments MPI applications and collects characterization logs fo the same
- **darshan-util**: utilities for analyzing the darshan log files

Darshan-runtime: Darshan-runtime has various responsibilities including intercepting IO function calls in the target application, extracting statistics, timing information and other IO-specific data. It also provides some general functionalities such as compression of log data to file for further evaluation. Figure 2 taken from ANL's website represents an example of POSIX instrumentation module and the Darshan runtime environment.

Darshan has a very modular design which allows developers to add extra instrumentation modules without breaking the existing toolkit. One such modular tool is the Darshan eXtended Tracing (DXT) [7]. DXT is a generic IO profiling tool that caters to almost all file systems. With Darshan 3.1.7 and above, DXT comes as a base module and can be enabled by setting an environment variable. It also comes with multiple useful functionalities like non-contiguous IO detection, data distribution analysis, IO bandwidth trace and

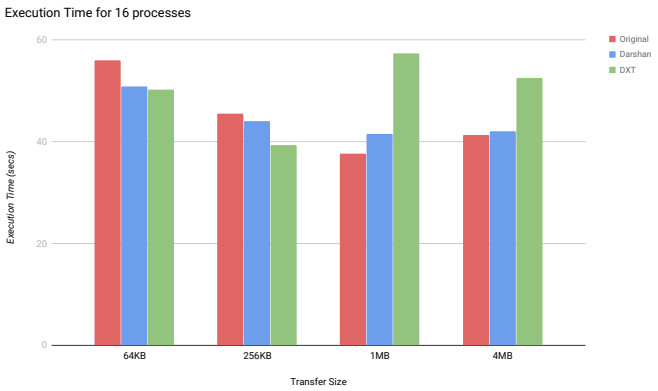

Figure 3: Execution time for an application without any instrumentation, with Darshan, and with DXT enabled for 16 processes.

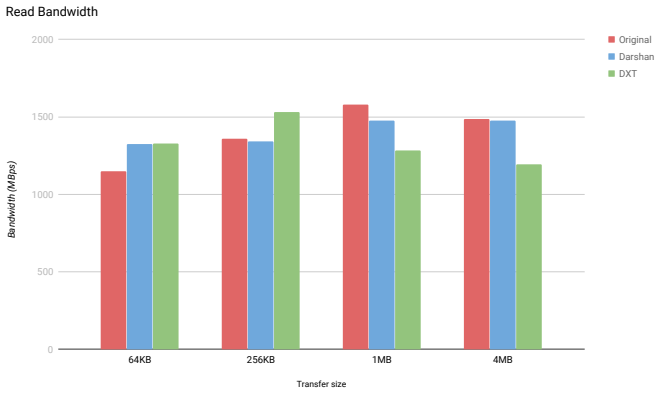

Figure 5: Read bandwidth for an application without any instrumentation, with Darshan, and with DXT enabled for 16 processes.

outlier detection.

Darshan-util: This is a collection of tools for parsing and summarizing the Darshan/DXT log files. This package can be installed independently of the test system and is a generic linux-compatible software.

3. BACKGROUND

IOR: Interleaved or Random [8] is a parallel IO benchmark. IOR provides performance testing on parallel file-systems using various interfaces and access patterns. IOR also uses MPI libraries for process synchronization. The *blockSize* in IOR is the total size that is transferred between two processes. The *transferSize* is the amount of data that is communicated in one go, totaling to the *blockSize*.

S3D-IO: [9] S3D-IO benchmarks the performance of PnetCDF method implementing the S3D combustion code.

4. DXT - OVERHEAD MEASUREMENT

We tested multiple benchmarks on both the HPC2010 and the CSE-cluster and the results, while consistent, were inconclusive. There can be multiple reasons for this like, the load on the system, the benchmarks used and the instrumentation tool.

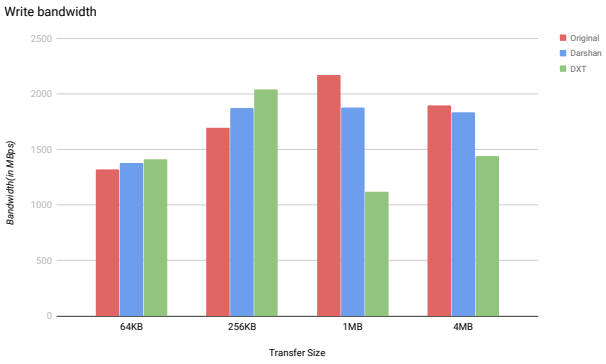

Figure 4: Write bandwidth for an application without any instrumentation, with Darshan, and with DXT enabled for 16 processes.

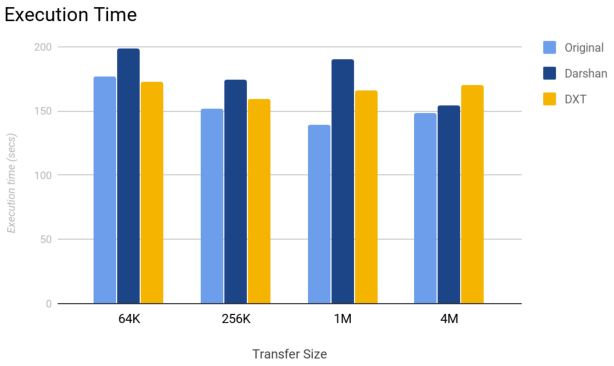

Figure 6: Execution time for an application without any instrumentation, with Darshan, and with DXT enabled for 64 processes.

Since we couldn't do anything about the load or the benchmark, we decided to test the instrumentation tool for any hidden overheads. In [7], authors have claimed that DXT has a very small overhead of just 1% in execution time. To test this claim, we perform an experiment with the same transfer sizes as used in [7]. On the courses queue, we can only run upto 128 processes (ppn:8), so we decided to perform the experiment for 16 processes and 64 processes. In Figure 3, we show the execution time for 16 processes, with transfer size as mentioned on x-axis. For 1MB and 4MB transfer size we can see overheads of upto 20 seconds, which is about 20% of the original execution time. Overall, for 16 processes we see about 14.6% increment in execution time.

In Figure 4, we also show the impact of DXT on the write bandwidth of the application. This particular metric was left out of consideration in the [7]. Our experiments show that DXT has a significant impact on the write bandwidth of the application, upto 1000 MB/s for 1MB transfer size and upto 500 MB/s for 4MB. Such results were not discussed in the paper and are not negligible at all to be ignored. Similarly, in Figure 5, we can see the read bandwidth decreasing by upto 300 MB/s for 1MB and 4MB transfer size.

To verify our results we also perform the experiment for 64 processes with the same transfer size. In Figure 6, we

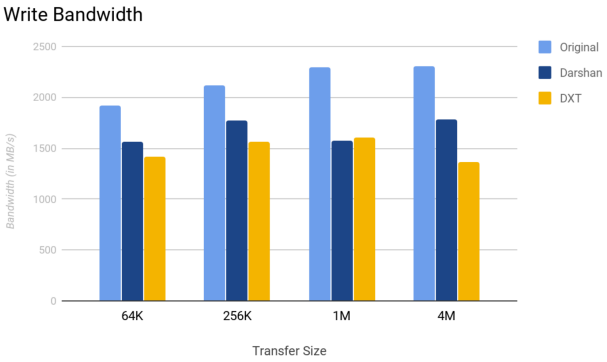

Figure 7: Write bandwidth for an application without any instrumentation, with Darshan, and with DXT enabled for 64 processes.

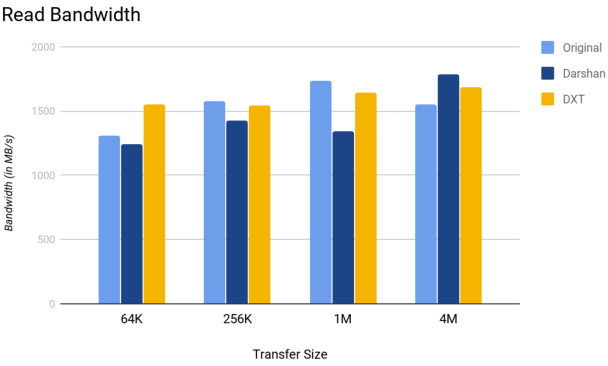

Figure 8: Read bandwidth for an application without any instrumentation, with Darshan, and with DXT enabled for 64 processes.

can see a trend similar to that of Figure 3. The execution time with DXT enabled increases by upto 20 seconds for 1MB and 4MB transfer size. The data shows an average execution time increment of about 9.4%. The degradation in write bandwidth ranges from about 500 to 1000 MB/s, as we can see in Figure 7. Read bandwidth on the other hand, shows an improvement of ranging from 100 to 250 MB/s in case of DXT when compared to the original read bandwidth.

We can think of a few reasons to justify this degradation in performance of the application in presence of DXT: (i) The system load and network congestion vary accross tests, (ii) DXT's algorithm is not optimized for smaller workloads and, (iii) The overhead data mentioned in the [7] is possibly from a previous release of the toolkit and newer addition and changes to the toolkit have increased its footprint.

For nullifying the effect of system load and network congestion we perform the test multiple time, and observe the same trend in IO performance of the benchmark. The other two reasons are quite plausible, and leave a scope for improvement.

5. I/O PROFILING EXPERIMENTS

5.1 Profiling IITK-CSE cluster

We use DXT as the our main tracing and profiling tool

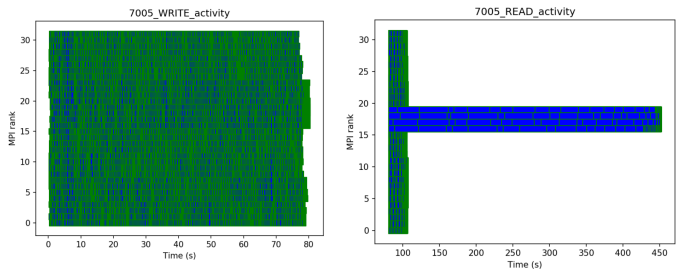

Figure 9: Read/write latency for 32 processes 8 nodes on the same switch on CSE-cluster for IOR

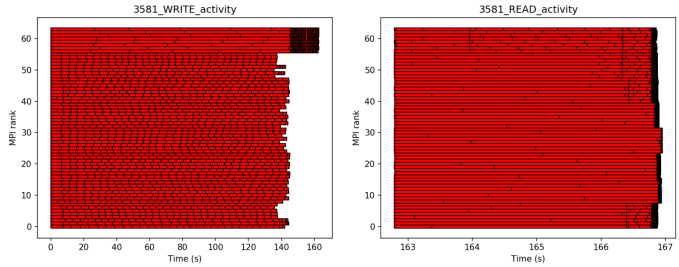

Figure 10: Read/write latency for 64 processes 8 nodes, 7 nodes on the same switch and one on different switch on CSE-cluster for IOR.

throughout this experiment. In this part of the experiment we run the benchmark on the IITK-CSE cluster which has a Netwrok File-System(NFS). The cluster has 2 switches. The nodes within the same switch are 2 hops(node -> switch -> node) apart and the nodes in different switches are 3 hops(node -> switch -> switch -> node) apart. Figure 9 shows the IO activity of the CSE cluster for 8 nodes on the same switch with a ppn of 4, blockSize of 16MB and transferSize of 1MB. The read behavior of processes with ranks 15-19 are quite abnormal. The writes from all processes take approximately 80 seconds and reads for most processes takes about 20 seconds, except for the four outliers. There is a read latency of over 350 seconds for these four processes. This could have been attributed to the load on that particular system, however, we were able to reproduce these observation for various other configurations.

Figure 11 shows the IO activity of CSE cluster for 8 nodes, one of which is on a switch-2. This can be clearly seen in the increased write-time of the first 8 process ranks in the plot. Surprisingly, the fact that the first node belongs to another switch doesn't affect the read time in the slightest.

Figure 11 shows the IO activity of CSE cluster for eight nodes two of which belong to switch-2. One can easily tell from the write-time plot that these are the first and the last nodes in the plot. Again we observe that only the write-time of processes belonging to different switches is affected, not the read-time. Another thing that can be read from the plots 11 and 10 is that while the processes on the primary switch (switch-1) are running the writes are slower for the processes on the secondary switch (switch-2), but they pick up speed once all other processes are finished writing.

5.2 Profiling HPC2010

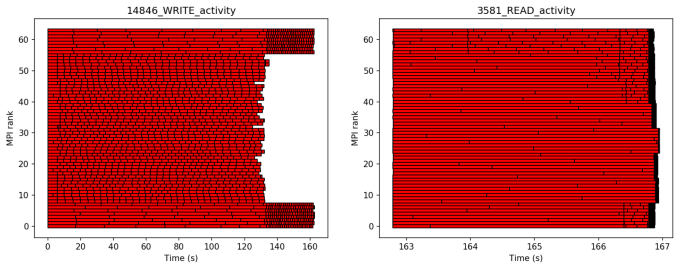

Figure 11: Read/write latency for 64 processes 8 nodes, 6 nodes on the same switch and two on different switch on CSE-cluster for IOR.

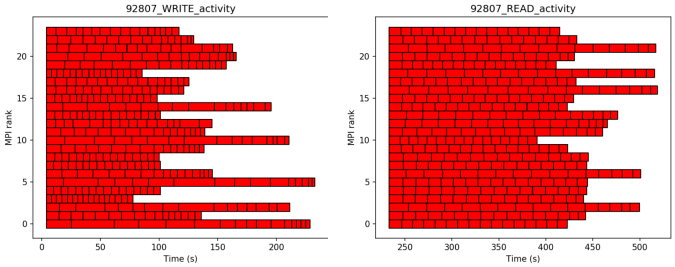

Figure 12: Read/write latency for 24 processes 3 nodes, on HPC2010 supercomputer cluster for IOR.

The HPC2010 supercomputer has a lustre file-system. HPC2010 also has 368 nodes segregated into enclosures of 16 nodes each. The communication between two nodes of the same enclosure is direct, whereas the communication between two different enclosures goes through three hops. For HPC2010, we test two benchmarks, IOR and S3D-IO. Figure 12 shows the IO activity, instrumented with DXT on HPC, of IOR benchmark for 3 nodes with a ppn of 8, blockSize of 1GB and transferSize of 1GB. The erratic read/write activity can be attributed to load on the particular node, network congestion or the benchmark itself.

In Figure 13, we show the IO activity of S3D-IO benchmark for 64 processes. The read/write latency is very uniform in this case. Multiple test runs with S3D-IO lead us believe that the erratic behavior of Figure 12 is due the way the IOR benchmark is written.

The plots of Figure 12 and 13 do not provide any conclusive relation between IO performance in HPC2010 and the topology of the supercomputer. To get some more idea about

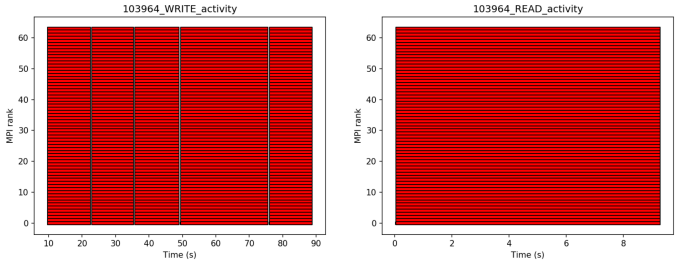

Figure 13: Read/write latency for 64 processes 8 nodes, on HPC2010 supercomputer cluster for S3D-IO.

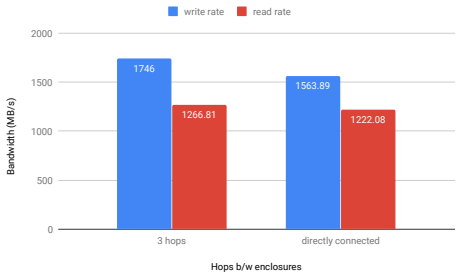

Figure 14: Average I/O bandwidth for processes running on same HPC enclosure and different enclosures.

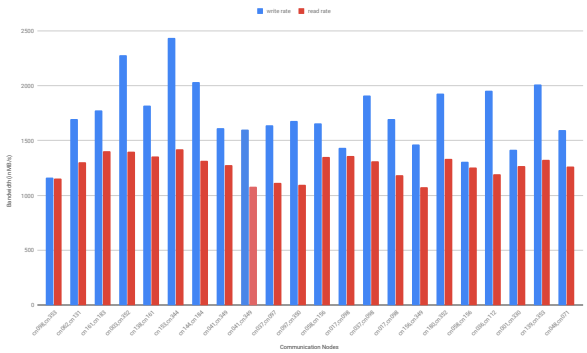

Figure 15: I/O bandwidth of processes from different enclosures, require 3-hops for communication.

the IO performance relation with the enclosure structure of nodes we perform the following experiments.

In this experiment, we measure the read/write bandwidth for IOR benchmark for nodes on the same enclosure and for two nodes on different enclosures. Figure 14 shows the average read/write bandwidth of communication with 3 hops (nodes on different enclosures) and communication on the same enclosures. The plot confirms our expectation, that the read/write bandwidth of nodes on the same enclosure is lower than that of the nodes on different enclosures, with 3 hops. The difference in bandwidth was not as pronounced as one would expect from direct and 3-hop communication.

Figures 15 and 16 show the bandwidth of various processes in the different enclosures and in same enclosures, respectively. The behavior of processes in both cases is somewhat similar and not and indicator of the difference that we would expect. Without a more in depth knowledge of the topology of the supercomputer nodes, it's difficult to comment on the relation between IO performance and topology.

6. CONCLUSION

In this project, we aimed to find some relation between the IO performance and topology of the CSE-cluster and HPC2010 supercomputer nodes. For HPC2010, we were able to show that the IO bandwidth for nodes in different enclosures is poorer than that of nodes in the same enclosure. We used DXT for tracing and instrumenting our benchmark results and on the way we digressed a bit to examine the overhead of DXT compared to Darshan and the original application. We were able to show some startling results about

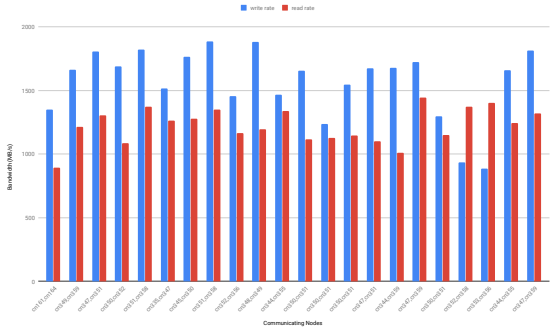

Figure 16: I/O bandwidth of processes from same enclosure, direct communication.

the overhead that the program execution time and bandwidth incur due to DXT. These results we not previously known, especially as [7] shows very different results from our own findings. There's some definite scope of improvement in DXT integration with processes, that can be taken up in future work.

7. APPENDIX

Member	% work	Work Done
Aditya Rohan	90	All the code and benchmarking; complete report
Anshul Vijayvergiya	10	Progress Slides

Table 1: Effect of stream direction on execution time (in cycles). T1: Thread 1 and T2: Thread 2.

8. REFERENCES

[1] S. J. Kim, S. W. Son, W.-k. Liao, M. Kandemir, R. Thakur, and A. Choudhary, "Iopin: Runtime profiling of parallel i/o in hpc systems," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pp. 18–23, IEEE, 2012.

[2] R. B. Ross, R. Thakur, *et al.*, "Pvfs: A parallel file system for linux clusters," in *Proceedings of the 4th annual Linux showcase and conference*, pp. 391–430, 2000.

[3] F. B. Schmuck and R. L. Haskin, "Gpfs: A shared-disk file system for large computing clusters.," in *FAST*, vol. 2, 2002.

[4] P. Schwan *et al.*, "Lustre: Building a file system for 1000-node clusters," in *Proceedings of the 2003 Linux symposium*, vol. 2003, pp. 380–386, 2003.

[5] Z. Abbasi, G. Gibson, B. Mueller, J. Small, M. Unangst, B. Welch, J. Zelenka, and B. Zhou, "Scalable performance of the panasas parallel file system," in *FASTâĂŹ08 Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pp. 17–33, 2008.

[6] A. N. Laboratory, "Darshan project," 2009.

[7] C. Xu, S. Snyder, V. Venkatesan, P. Carns, O. Kulkarni, S. Byna, R. Sisneros, and K. Chadalavada, "Dxt: Darshan extended tracing," tech. rep., Argonne National Lab.(ANL), Argonne, IL (United States), 2017.

[8] HPC, "Ior," 2019.

[9] W.-k. Liao and A. Choudhary, "Dynamically adapting file domain partitioning methods for collective i/o based on underlying parallel file system locking protocols," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, p. 3, IEEE Press, 2008.