Diagnosing Parallel I/O Bottlenecks in HPC Applications

Peter Harrington
Baskin School of Engineering
University of California Santa Cruz
Email: pharring@ucsc.edu

***Abstract*—High-Performance Computing (HPC) applications are generating increasingly large volumes of data (up to tens of PBs), which need to be stored in parallel to be scalable. Parallel I/O is a significant bottleneck in HPC applications, and is especially challenging in Adaptive Mesh Refinement (AMR) applications because the structure of output files changes dynamically during runtime. Data-intensive AMR applications run on the Cori supercomputer at NERSC show variable and often poor I/O performance, but diagnosing the root cause remains challenging. Here we analyze logs from multiple levels of Cori's parallel I/O subsystems, and find bottlenecks during file metadata operations and during the writing of file contents that reduced I/O bandwidth by up to 40x. Such bottlenecks seemed to be system-dependent and not the fault of the application. Increasing the granularity of file-system performance data will help provide conclusive causal relationships between file-system servers and metadata bottlenecks.**

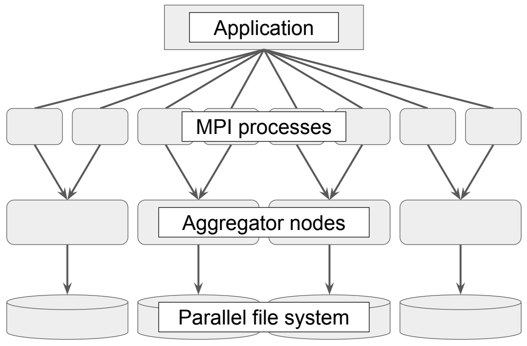


Fig. 1. A schematic of the flow of application data through the different components of a typical HPC parallel I/O system.

I. Introduction

High-performance computing (HPC) systems are of paramount importance to countless scientific and engineering fields, and have undergone great advancements in scale and capacity over the previous decades. This has brought about a significant increase in the complexity and parallelism of supercomputer architectures as well as the amount of data generated by HPC applications. With such large data volumes, parallel I/O has become a crucial component of modern HPC systems, but also an increasingly significant performance bottleneck. Achieving peak parallelism and performance in I/O will be an important step to building any feasible exascale computing system [1]. Identifying parallel I/O bottlenecks and their cause will improve efficiency and scalability at both the application level as well as the system level.

An HPC system uses multiple layers of software and hardware to implement parallel I/O. Data is transferred between compute nodes via some parallel programming framework, such as Message Passing Interface (MPI) or OpenMP, but if the data is to remain accessible after application execution completes, it must be written onto a parallel file system (e.g., Lustre or GPFS). A schematic example of a typical HPC parallel I/O system is given in Fig. 1. Performance profiling tools exist for various individual levels of the underlying parallel I/O subsystems, but comprehensive analysis of I/O performance is challenging due to the large variance in the amount, availability, and granularity of performance data. Furthermore, because all concurrent applications on an HPC system share the same parallel I/O resources, interference from external system activity introduces another source of variance for any individual application's I/O performance. Thus, while the importance of understanding and improving parallel I/O performance is well-known, the tools for comprehensive analysis of parallel I/O performance remain elusive.

II. Background & Related Work

The majority of previous parallel I/O performance analyses have focused on studying individual levels of a parallel I/O system, usually either at the application level or file-system level. Application-level I/O tracing typically consists of collecting statistics on or directly tracking an individual application's I/O calls (which are made through interfaces like POSIX-IO and MPI-IO). Many tools have been built to conduct application-level I/O studies, and have successfully been used to identify poor I/O performance patterns. Examples of such tools include Darshan [2], Integrated Performance Monitoring [3], Trace+ [4], RIOT IO [5], ScalaIOTrace [6], TraceFS [7], IOSig [8], and IORecorder [9]. At the other end of the parallel I/O stack, several file-system performance monitoring tools have been developed to collect information on the I/O performance of the servers within parallel file-systems like Lustre [10] and GPFS [11].

Analyzing parallel I/O performance using the aforementioned tools can be labor-intensive, and only gives information about a specific level of the underlying parallel I/O subsystems. Combining application-level and file-system-level I/O performance data is even more challenging, and such integrated analysis has been done only a few times and with limited scope. Lu & Shen [12] used multi-layer event trace analysis to find degraded I/O performance, and attributed it to issues with the asynchronous I/O implementation in the GNU C runtime library as well as frequent disk seek/rotation operations from concurrent MPI-IO accesses. In a different work, Xu et al. [13] developed LIOProf as an effort towards bridging the gap between application-level parallel I/O libraries, middleware, and the Lustre file system. They successfully used it to identify issues with MPI-IO collective reads as well as the overhead of the HDF5 I/O library when used on a Lustre file-system.

III. Approach & Uniqueness

Towards developing a general framework capable of scalable and holistic parallel I/O performance analysis, this work combined performance data from multiple layers of a parallel I/O implementation on a production HPC system. In particular, application-level performance logs and file-system performance logs were simultaneously analyzed with the goal of finding application-side I/O bottlenecks and studying the corresponding file-system behavior during periods of degraded performance.

The HPC system analyzed in this work was the Cori supercomputer, at the National Energy Research Scientific Computing Center (NERSC). Cori is a Cray XC40 system with a theoretical peak performance of 31.4 PFlops and a peak file-system I/O bandwidth of 700 GB/s. Application-level POSIX and MPI I/O traces were generated with the Darshan library, with extra detail provided for some application runs by the Darshan Extended Tracing tools [16]. File-system performance data was logged by the Lustre Monitoring Tool (LMT), which gathers performance data for the Object Storage Servers/Targets (OSSs/OSTs) and Metadata Servers/Targets (MDSs/MDTs) of the Lustre parallel file system. We focus on the I/O behavior of two sample HPC AMR applications, `HyperCLaw` [14] and `Chombo` [15].

Parsing and analyzing the logs from large numbers of I/O intensive application runs can be difficult and time-consuming, so Apache Spark [17] was used to both parallelize and simplify the analysis workflow. Doing so stored performance data from the application level and the file-system level into data structures that could be queried for the I/O performance data specific to a certain file, rank, time after program start, or individual file-system OSTs or MDSs. In this way, bottlenecks appearing as anomalously long wait times or slow I/O speeds on the application side could be cross-checked with the external load on file-system servers. A schematic diagram of this workflow is given in Fig. 2.

As detailed in the previous section, the bulk of previous attempts to characterize parallel I/O performance bottlenecks have focused on individual levels of a parallel I/O stack, with the exception of a few targeted efforts that combined application-level or file-system-level performance data with some information from another I/O layer. The workflow presented here allows the simultaneous analysis of these two key levels of parallel I/O systems, and offers a basis for a more general framework allowing holistic parallel I/O analysis.

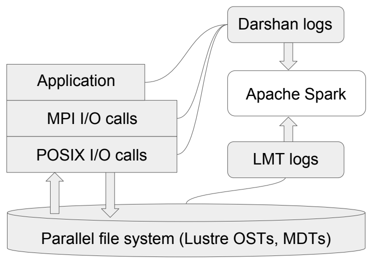


Fig. 2. A diagram of the analysis workflow. Darshan logs collect application-level I/O traces, while the LMT logs collect file-system usage data. Logs are parsed and analyzed in parallel using Apache Spark.

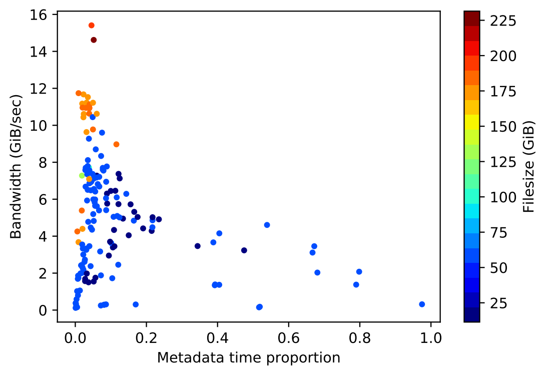


Fig. 3. The write bandwidth relative to the proportion of total I/O time spent in metadata operations, for all 79 runs of `HyperCLaw`. Bottlenecks happened when metadata operations dominated total I/O time (bottom right) and when writing of file contents dominated total I/O time (bottom left). Some outlying cases existed where bottlenecks in both components led to a roughly equal contribution to total I/O time from each (bottom center).

IV. Experimental Results

Applying the analysis workflow to a set of 79 `HyperCLaw` runs detected several runs with significant slowdowns that caused as much as a 40x reduction in I/O bandwidth. These occurred when metadata operations or data write operations, and sometimes both, took much longer than usual and dominated the I/O time of the application. The relationship between write bandwidth and amount of I/O time spent in metadata operations can be seen in Fig. 3. Figure 4 shows the slowdown ratio of each run with respect to the normal bandwidth of runs with similar parameters. Other runs with identical settings to the bottlenecked cases were observed to run perfectly fine, so it appears that slowdowns during metadata operations and data writing were not the fault of the application.

File-system logs sampled activity every 5 seconds, and MDS CPU load was only measured for the primary MDS but not the four others. This limited granularity obscured the relationship between file system traffic and application-side I/O performance in bottlenecked runs. For example, even in the worst-performing run that saw a bottleneck in both metadata

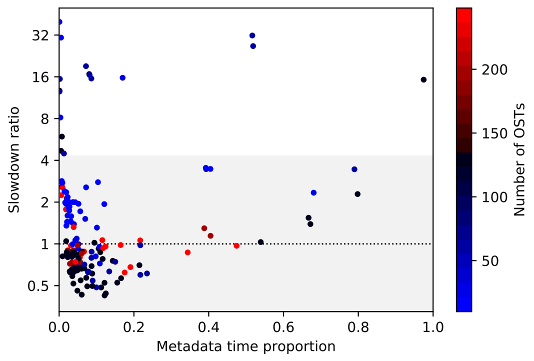


Fig. 4. The slowdown ratio of each run's actual bandwidth relative to normal bandwidth (log scale), with the 90% range shown in gray. Besides runs with too few (blue) or too many (red) OSTs, most runs with a 2x or more slowdown had file system bottlenecks.

and data write operations for both files, only one of the file write periods coincided with elevated file system traffic (see Fig. 5).

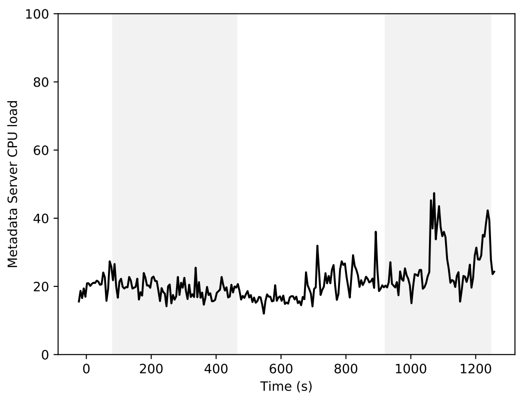
An additional bottleneck was observed in the applications with HDF5 collective writes, showing several trailing writes occurring after the main write output phase (see Fig. 6). These writes took up to 5.5 seconds, despite writing less than 5 kB of data (a small fraction of the total output), and were highly inefficient relative to the write bandwidth during the main write phase. Their duration increased as the file size and number of processes increased, so their behavior is not scalable. Examination of the MPI ranks and OST data indicated that these trailing writes may have been updating some HDF5 metadata within the file after AMR grid refinement.

V. Conclusions

While the largest slowdowns in these application runs occurred during data writing, there were also significant slowdowns during metadata operations that seemed not to be the application's fault. Most runs (90%) spent within 23% of total I/O time in metadata operations, and anything higher resulted in degraded performance. As traffic on HPC file systems continues to rise, it is likely that parallel I/O bottlenecks arising during metadata operations will become more frequent and severe.

Repeating this analysis with higher-resolution performance data for file-system activities should enable pinpointing such metadata bottlenecks to individual servers. During runs that did not experience any bottlenecks, many periods of metadata or data write operations took between 0.5-5 seconds (less than the time-resolution setting of the file-system performance monitoring tools), so knowing the load on file-system servers over shorter intervals is essential to finding correlations between file-system traffic and application-specific I/O performance.

The analysis workflow implemented in this study has demonstrated its ability to find parallel I/O bottlenecks or other inefficient I/O behavior, and suggest potential root causes for



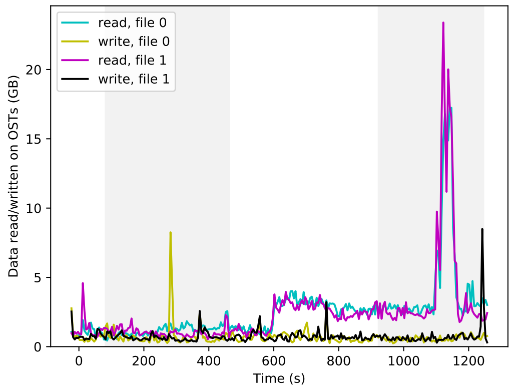


Fig. 5. The CPU load of the primary metadata server (top) and bytes transferred across each file's OSTs (bottom) for the two files of a run where both metadata activities and data writing were bottlenecked. The write period of each file is shaded in gray.

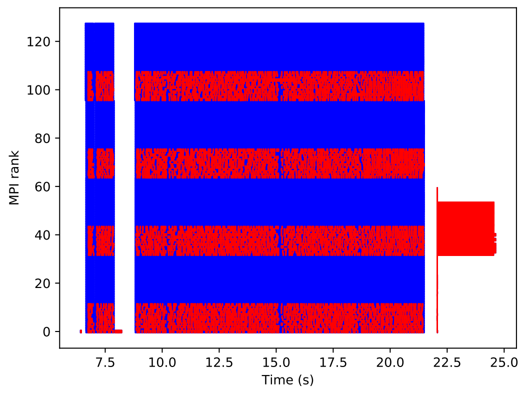


Fig. 6. MPI (blue) and POSIX (red) writes for each of the 128 MPI processes during a 104 GB `Chombo` run. Trailing writes occur near 22 seconds.

those bottlenecks. Because it combines data from two key levels – the application level and the file-system level – it can be used as a basis for developing a comprehensive, holistic parallel I/O analysis framework. Such a framework would be an indispensable tool for diagnosing parallel I/O bottlenecks within the current paradigm and for aiding in the design of next-generation parallel I/O systems.

VI. Contributions and Further Work

After initial submission of this work, some of the results, as well as the general workflow used, have contributed to or been the basis of further related work. The tools used to create plots of I/O activity such as that shown in Figure 6 have been enhanced slightly and added to the recent Darshan 3.1.6 distribution as `dxt_analyzer`.

The inefficient I/O behavior observed during HDF5 collective writes, an example of which is in Figure 6, has been fixed in the collective metadata handling feature of HDF5. This feature improves I/O performance by $\sim 10\%$ on average for all cases. An example of the performance improvements for `Chombo` is shown in Fig. 7.

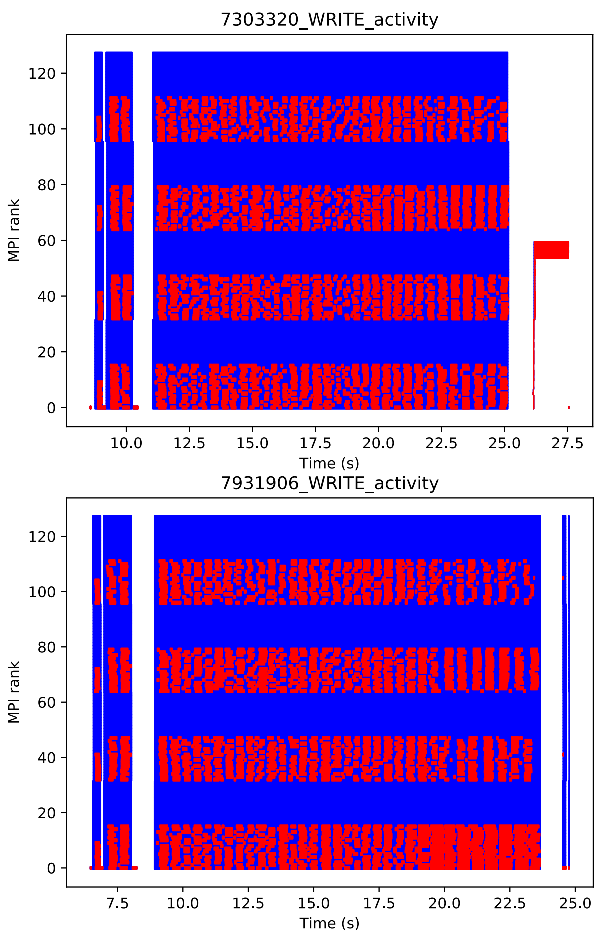


Fig. 7. The I/O performance of `Chombo` runs before (top panel) and after (bottom panel) the inefficient I/O behavior found by this work was corrected.

VII. Acknowledgements

I would like to thank Alex Sim, John Wu, William Yoo, and Surendra Byna at Lawrence Berkeley National Laboratory (LBNL) for their guidance on this work, Vincent Beckner and Ann Almgren at LBNL for the AMReX HyperCLaw HPC application, Brian Van Straalen at LBNL for the Chombo HPC application, and Shane Snyder and Phil Carns at Argonne National Laboratory (ANL) for their support in the Darshan tool. This work utilized resources of the National Energy Research Scientific Computing Center (NERSC). This work was supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02- 05CH11231. This work was supported in part by the U.S. Dept. of Energy, Office Science, Office of Workforce Development of Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internship (SULI) program.

References

[1] Kogge, P. et al. *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*. Univ. of Notre Dame, CSE Dept. Tech. Report TR-2008-13. (2008)
[2] Carns, P., et al. "Understanding and Improving Computational Science Storage Access through Continuous Characterization". *Trans. Storage*, 7, 3. (2011)
[3] Wright, N. J., Pfeiffer, W., & Snavely, A. "Characterizing Parallel Scaling of Scientific Applications using IPM". *Proceedings of the 10th LCI International Conference on High-Performance Clustered Computing* (2009).
[4] Mesnier, M. P., et al. "Trace: Parallel Trace Replay with Approximate Causal Events". *Proceedings of the 5th USENIX Conference on File and Storage Technologies* (2007).
[5] Wright, S.A., et al. "Parallel File System Analysis Through Application I/O Tracing". *Comput. J.* (2012)
[6] Vijayakumar, K., et al. "Scalable I/O Tracing and Analysis". *Proceedings of the 4th Annual Workshop on Petascale Data Storage*. (2009)
[7] Aranya, A., Wright, C. P., and Zadok, E. "TraceFS: A File System to Trace Them All". *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*. (2004)
[8] Yin, Y., et al. "Boosting Application-Specific Parallel I/O Optimization Using IOSIG". *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. (2012)
[9] Luu, H., et al. "A multi-level approach for understanding I/O activity in HPC applications". *2013 IEEE International Conference on Cluster Computing* (2013).
[10] Wartens, H. C. M. & Garlick, J. "LMT - The Lustre Monitoring Tool". https://github.com/chaos/lmt/wiki/. Developed at Lawrence Livermore National Lab. (2010)
[11] *IBM GPFS Monitoring and Tools*, https://goo.gl/7lOzhr. (2016)
[12] Lu, P., & Shen, K. "Multi-Layer Event Trace Analysis for Parallel I/O Performance Tuning". *2007 International Conference on Parallel Processing*. (2007)
[13] Xu, C., et al. "LIOProf: Exposing Lustre File System Behavior for I/O Middleware". (2016).
[14] Welcome, M., et al. "Performance Characteristics of an Adaptive Mesh Refinement Calculation on Scalar and Vector Platforms". *Proceedings of the 3rd Conference on Computing Frontiers*. (2006)
[15] Adams, M., et al. "Chombo Software Package for AMR Applications - Design Document". Lawrence Berkeley National Laboratory Technical Report LBNL-6616E. (2015)
[16] Xu, C., et al. "DXT: Darshan eXtended Tracing". (2017)
[17] Zaharia, M., et al. "Spark: Cluster Computing with Working Sets". *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. (2010)