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 hundreds of TBs), 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 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 application's fault. Increasing the granularity of file-system performance data will help provide conclusive causal relationships between file-system servers and metadata bottlenecks.**

I. Introduction

High-performance computing systems have undergone great advancements in scale, efficiency, and capacity over the previous decades, which has brought about a significant increase in the complexity and parallelism of modern supercomputer architectures. In such HPC systems, achieving peak parallelism and performance in parallel I/O will be an important step in building any feasible exascale computing system [1]. Finding parallel I/O bottlenecks and identifying 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. System logs exist for various levels of this stack, but comprehensive analysis of I/O performance is challenging due to the size, granularity, and varying sources of performance data.

II. Methods

The methodology employed by this work focuses on integrating performance logs from two key levels of Cori's parallel I/O subsystems: application-level POSIX and Message Passing Interface (MPI) I/O traces, coming from the Darshan library [2], and performance data logged by the Lustre Monitoring Tool (LMT) [3], a monitoring tool for the Object Storage Targets (OSTs) and Metadata Servers (MDSs) of the Lustre parallel file system. We focus on the I/O behavior of two sample HPC AMR applications, `HyperCLaw` [4] and `Chombo` [5].

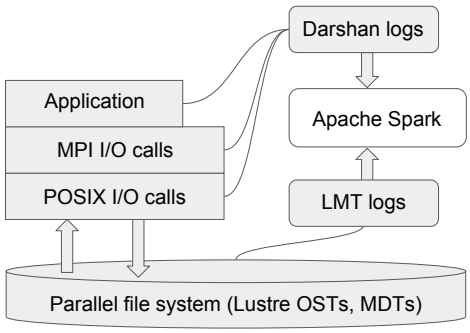

Fig. 1. A schematic of the parallel I/O software stack analyzed in this work. The Darshan logs collect application-level I/O traces, while the LMT logs collect file-system usage data. The data from both is analyzed in parallel using Apache Spark.

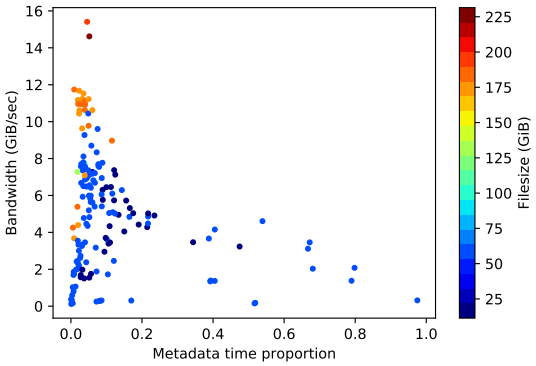

Fig. 2. The write bandwidth relative to the proportion of total I/O time spent in metadata operations, for all 79 runs of `HyperCLaw`. Bottlenecks happened when metadata operations dominated total I/O time (bottom right) and when writing of file contents dominated total I/O time (bottom left). Some outlying cases existed where bottlenecks in both components led to a roughly equal contribution to total I/O time from each (bottom center).

The I/O performance logs from large, highly parallel applications with heavy I/O traffic are large in size and unwieldy to process. To efficiently analyze all of this information, Apache Spark [6] was used to accelerate log-parsing and make I/O activity for specific times and MPI ranks easily accessible. A schematic diagram of this workflow is given in Figure 1.

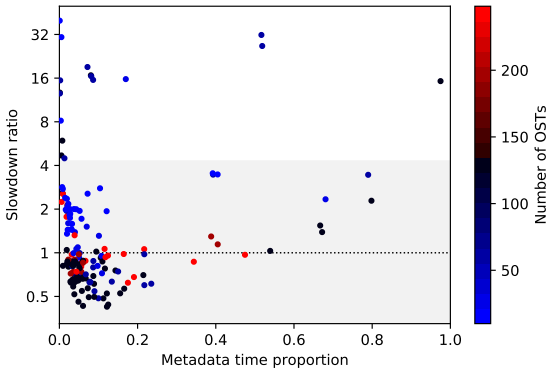

Fig. 3. The slowdown ratio of each runs actual bandwidth relative to normal bandwidth (log scale), with the 90% range shown in gray. Besides runs with too few (blue) or too many (red) OSTs, most runs with a 2x or more slowdown had file system bottlenecks.

III. Experimental Results

Even with optimized file system stripe settings, some runs showed significant slowdowns that caused as much as a 40x reduction in I/O bandwidth. These bottlenecks 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 Figure 2. Figure 3 shows the slowdown ratio of each run with respect to the normal bandwidth of runs with similar parameters. Because other runs with identical settings to the bottlenecked cases were observed to run perfectly fine, it appears that slowdowns during metadata operations and data writing were not the fault of the application.

The file-system logs sampled activity once every 5 seconds, and MDS CPU load was only measured for the primary MDS and not the four others. Consequently, 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 operations and data writing during each file's writing period, only one of the writing periods coincided with elevated file system traffic (see Figure 3).

We also observed an additional bottleneck in the applications with HDF5 collective writes, showing several trailing writes that occurred after the main write output phase (see Figure 5). The duration of these writes lasted up to 5.5 seconds, despite only writing less than 5 kB of data (a small fraction of the total output), and increased when the total file size and number of processes increased. Such behavior is inefficient and does not scale.

IV. 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

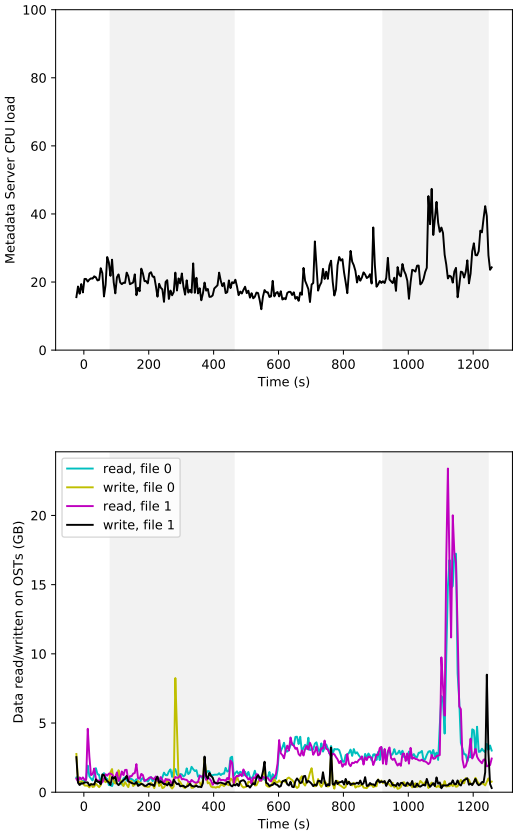

Fig. 4. The CPU load of the primary metadata server (top) and bytes transferred across each file's OSTs (bottom) for the two files of a run where both metadata activities and data writing were bottlenecked. The write period of each file is shaded in gray.

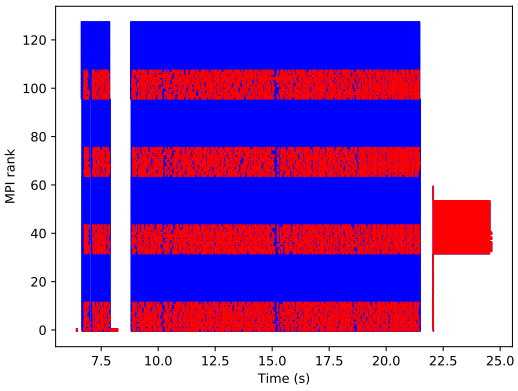

Fig. 5. MPI (blue) and POSIX (red) writes for each of the 128 MPI processes during a 104 GB `Chombo` run. Trailing writes occur near 22 seconds.

I/O time in metadata operations, depending on the amount data they wrote, and anything higher than this usually resulted in degraded performance, as can be seen in Figures 2 and 3. As the 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.

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 operations or data write operations took between 0.5-5 seconds. Therefore, knowing the load on file-system servers over intervals shorter than five seconds (as was the case in this work) is essential to finding correlations between file-system traffic and application-specific I/O performance. While activity on the primary metadata server is indicative of system load, knowing the load on each of the secondary metadata servers will also give a more comprehensive picture.

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] Wartens, H. C. M. & Garlick, J. LMT - The Lustre Monitoring Tool. https://github.com/chaos/lmt/wiki/. Developed at Lawrence Livermore National Lab. (2010)

[4] 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)

[5] Adams, M., et al. "Chombo Software Package for AMR Applications - Design Document". Lawrence Berkeley National Laboratory Technical Report LBNL-6616E. (2015)

[6] Zaharia, M., et al. "Spark: Cluster Computing with Working Sets". Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing. (2010)