Diagnosing Parallel I/O Bottlenecks in HPC Applications

Peter Harrington[1], Wucherl Yoo (advisor)[2], Alexander Sim (advisor)[2], Kesheng Wu (advisor)[2]
[1]University of California Santa Cruz, [2]Lawrence Berkeley National Laboratory

Abstract

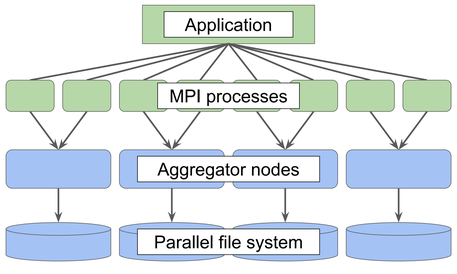
High-Performance Computing (HPC) applications are generating increasingly large volumes of data, which need to be stored in parallel to be scalable. Parallel I/O is a significant bottleneck in HPC applications, and is especially challenging in Adaptive Mesh Refinement (AMR) applications because the structure 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.

Research Question

What is causing performance bottlenecks in HPC applications?

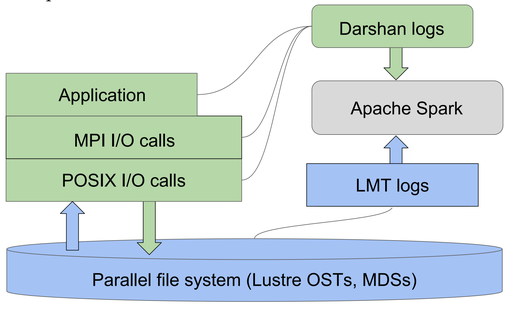
Background & Approach

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.



The flow of application data through Cori's parallel I/O subsystems

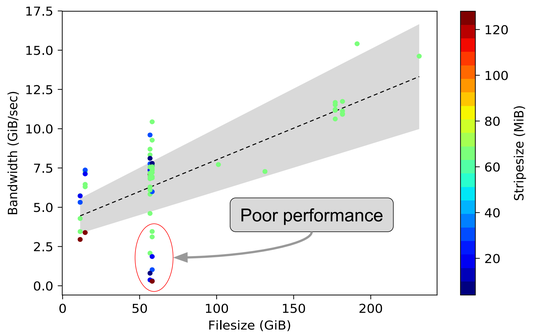
We focus on the I/O behavior of two sample HPC AMR applications, `HyperCLaw` and `Chombo` , and use Darshan to track application-level I/O activities and Lustre Monitoring Tool (LMT) to track Lustre parallel file system activities on Cori's 248 Object Storage Targets (OSTs). Data was processed in parallel using Apache Spark.



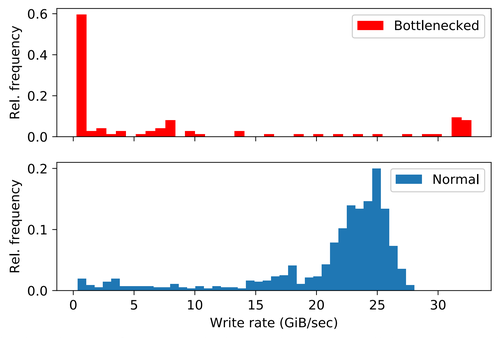
The workflow used to analyze I/O performance.

Locating and Diagnosing Bottlenecks

Even with optimized file system stripe settings, some runs showed significant slowdowns that caused as much as a 40x reduction in I/O bandwidth.

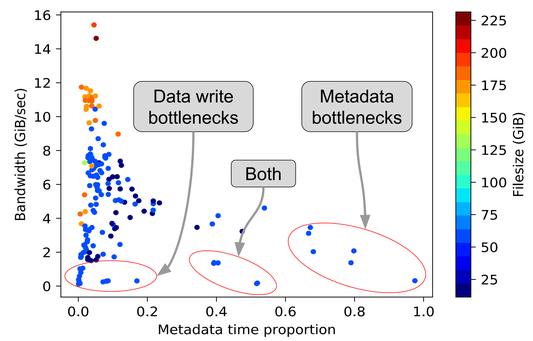


The write bandwidth of `HyperCLaw` at different file sizes, for the best performing stripe count (128 OSTs per file).

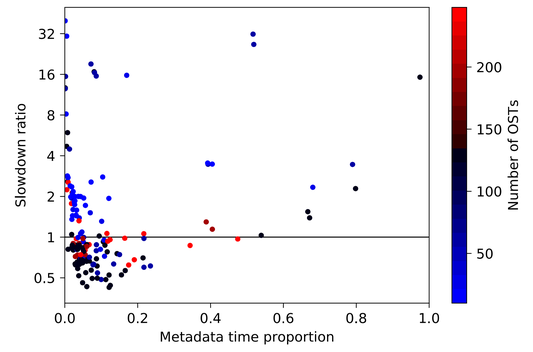


The distribution of write speeds during the main parallel write phase of a run that was bottlenecked during data write (top) and normal run (bottom).

During bottlenecked runs, either metadata operations or data write operations took much longer than usual, and in some cases both did.

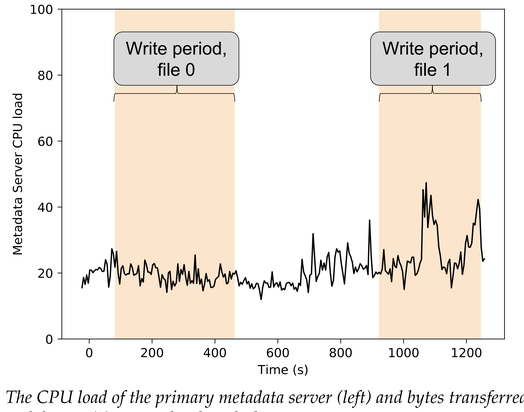


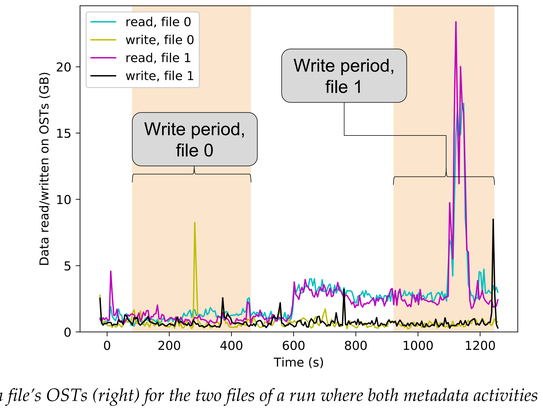
The relationship between time spent in file metadata operations (relative to total I/O time) and bandwidth for 79 runs of `HyperCLaw`.



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

Bottlenecked performance sometimes corresponded with elevated levels of file system traffic, but the limited granularity of the traffic data could not conclusively assign responsibility to specific servers.

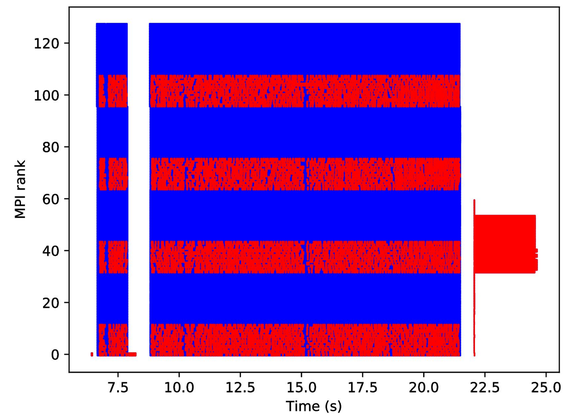




The CPU load of the primary metadata server (left) and bytes transferred across each file's OSTs (right) for the two files of a run where both metadata activities and data writing were bottlenecked.

The file-system logs sampled activity once every 5 seconds, and metadata server CPU load was only measured for the primary MDS and not the four others. Even in runs where both files experienced significant metadata and data write bottlenecks (such as the one pictured above), this limited granularity obscured the relationship between file system traffic and application-side I/O performance.

Additional Observations



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.

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. This behavior degraded two important performance properties:

- **Efficiency**
  Duration of trailing writes lasted up to 5.5 seconds, despite only writing less than 5 kB of data (a small fraction of the total output).
- **Scalability**
  Time taken by the writes increased when the total file size and number of processes increased, so such behavior does not scale.

Conclusions

- **Slowdowns that reduce I/O bandwidth by up to 40x can occur during metadata operations and/or data writing, and are not the application's fault.**
- **For 90% of runs, metadata operations consumed up to 23% of the total I/O time. Higher proportions of time spent in metadata operations resulted in bottlenecks.**

Future Work

- **Finer-granularity file-system measurement data will be helpful in ascribing metadata bottlenecks to specific servers within the file-system.**
- **Determining the source of trailing writes observed after the main collective I/O phase will help improve application performance and scalability.**

Acknowledgements

I would like to thank Surendra Byna at LBNL for his guidance on HPC I/O insights, Vincent Beckner and Ann Almgren at LBNL for the AMReX `HyperCLaw` HPC application, and Brian Van Straalen at LBNL for the `Chombo` HPC application. 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.