Reading from a Write Optimized Parallel File System Under Highly Concurrent Shared File Workloads

Adam Manzanares (Postdoc)[1], Milo Polte (Student)[2], Meghan Wingate[1], John Bent[1], and Garth Gibson[2]

[1]Los Alamos National Laboratory
[2]Carnegie Mellon University

The Parallel Log Structured File System (PLFS) was designed at the Los Alamos National Laboratory (LANL) to achieve high performance on highly concurrent shared file workloads. PLFS is used as a virtual file system interposed between the application and an underlying parallel file system, which we refer to as the physical store. Large parallel applications (on the order of thousands to hundreds of thousands of processes) use PLFS for concurrent writing to a shared file. PLFS has demonstrated performance improvements on this workload by one to two orders of magnitude on three of the major parallel file systems (PanFS, Lustre, and GPFS).

To achieve this improved performance, PLFS reorganizes the applications writes. As the application writes to a shared logical file, PLFS, instead of applying the writes to the specified offsets within a single file on the physical store, creates a single log-structured file on the physical store for each process writing to the shared logical file. To reconstruct the file for reading, PLFS also maintains index files on the physical store, mapping offsets within the logical file to their position within the log-structured files on the physical store.

This transformation of a shared logical file into independent log-structured components is the key technique that improves the write bandwidth of parallel applications, but this improved performance comes at the cost of complexity for read operations. In this work we plan to investigate optimizations that address these challenges and ensure that the read performance of files read through PLFS matches that of files stored directly on the physical store. There are at least three challenges in reading PLFS files. The first is that the data is stored in a log-structured fashion across multiple files on the physical store such that some patterns of reads might cause multiple seeks in multiple physical files. The second is that the cost to aggregate the multiple physical index files needed to reconstruct the logical file increases quadratically with the size of the parallel job that wrote the file. Finally, maintaining a consistent view of a file that is opened simultaneously for both read and write by multiple concurrent processes in PLFS introduces performance bottlenecks.

The traditional solution to the first challenge is to flatten the log-structured file(s) into a single contiguous file. However, the flattening cost can be expensive for large files in a parallel file system. We propose various techniques to avoid flattening the file while achieving the read bandwidth possible were it flattened. We will examine several read workloads, some which actually perform worse on a flattened file compared to a file in the PLFS format. Finally, we intend to show that some read workloads of a PLFS file fail to fully exploit the underlying parallel file system and how we can reintroduce parallelism to improve bandwidth.

The second challenge is particularly difficult when multiple readers concurrently open a PLFS file. In this case, each reader must aggregate the same index by reading the same set of index files from the physical store. Note that there are three current methods to access PLFS files: one, through a normal POSIX file system mount point (via FUSE); two, through the MPI-IO API already used by many parallel applications; and three, through the PLFS API. This challenge can be addressed when PLFS is accessed through the MPI-IO interface since it passes MPI communicators to PLFS such that PLFS, which runs otherwise as independent daemons, can work cooperatively in parallel. We examine several techniques exploiting these communicators to reduce the workload on the physical store. We also examine how the functionality provided by these communicators can be replicated when the communicators are not available.

Finally, we discuss the particularly difficult challenge introduced by simultaneous readers and writers of a PLFS file and how independent processes can synchronously maintain persistent index information and at the same time avoid serialization on the underlying parallel file system.