BurstFS: A Distributed Burst Buffer File System for Scientific Applications

Teng Wang[†], Kathryn Mohror[‡], Adam Moody[‡], Weikuan Yu[†], Kento Sato[‡]
[†] Florida State University [‡] Lawrence Livermore National Lab
{twang, yuw}@cs.fsu.edu {kathryn, moody20, sato5}@llnl.gov

I. Introduction

The computing power on leadership-class supercomputers has been growing at an unprecedented rate, and is projected to reach exascale in the near future. This growth allows application scientists to solve more complex scientific problems at larger scales, consequently producing gigantic volumes of data for checkpoint/restart, data analysis and visualization. These datasets challenge the underlying parallel file system (PFS), and as a result, extending the PFS to handle the I/O workload from exascale computing has become a prohibitive endeavor.

Recently, researchers have been exploring a new storage architecture that deploys burst buffers (composed of DRAM and SSD) between the compute nodes and the backend PFS [7], [8]. Scientific applications can store and retrieve intermediate data from burst buffers (BB) with high bandwidth. A node-local BB architecture, which locates BB on the individual compute nodes, has been adopted by several computing facilities, such as the Catalyst [1] and Hyperion [2] clusters from Lawrence Livermore National Laboratory, and Oak Ridge National Laboratory's next-generation supercomputer Summit [3]. Despite its outstanding scalability, there is still a lack of corresponding software solutions to exploit this new storage architecture. While prior efforts have researched the use of BB for checkpointing [6] and scientific analytics [10], there is still a gap for general scientific I/O workloads.

In this study, we propose BurstFS, a distributed BB file system, to exploit this architecture and provide scientific applications with high and scalable performance for bursty I/O requests. Our initial evaluation demonstrated that BurstFS improved bandwidths by 1.83× and delivered good scalability.

II. BurstFS Architecture

Figure 1 shows the BurstFS architecture. BurstFS is built on top of CRUISE [6], a file system designed to support checkpoint/restart workloads. CRUISE intercepts applications' POSIX function calls at link time and leverages both DRAM and SSD storage. Each process writes its data to the local storage, which makes CRUISE an efficient and scalable solution. CRUISE is mainly designed for N-N writes, where N processes each write a separate file, and is less efficient for N-1 writes, where N processes write to a single shared file. BurstFS overcomes this limitation by following a log-structured approach like in PLFS [4]; it transforms the N-1 pattern into an N-N pattern and builds the corresponding index for future data retrieval. One major challenge is that

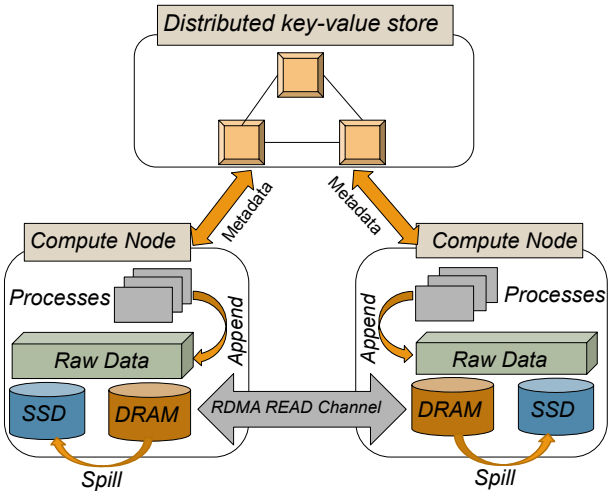
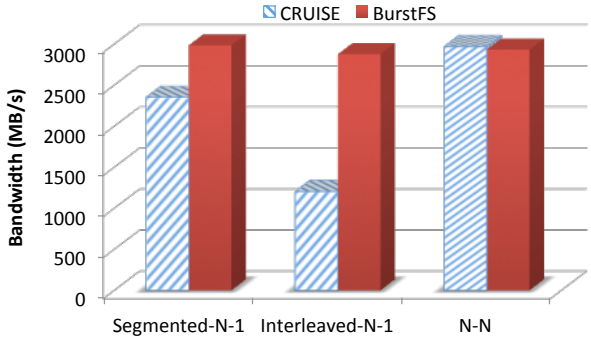


Fig. 1: BurstFS Architecture

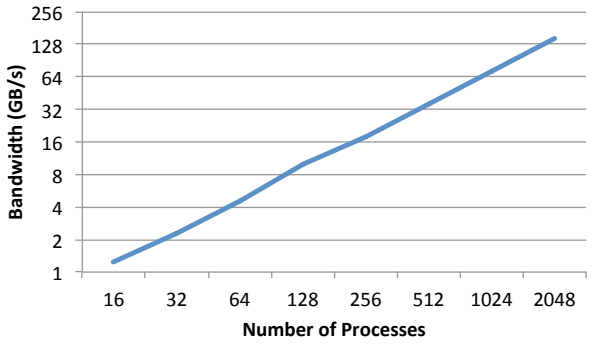
both indexing and the enormous number of aggregated files can result in substantial metadata. BurstFS spreads this metadata workload over a distributed key-value store. Moreover, BurstFS allows one process to retrieve the data from the remote node following an RDMA read, which is an ongoing extension on top of CRUISE for supporting local reads. Since individual compute nodes are generally dedicated to a single parallel job at a time on HPC systems, BurstFS's lifetime is associated with the individual job. At the end of each job, all data are sanitized. Data persistence on PFS can be achieved using existing approaches [4], [9].

III. Preliminary Results

We built an initial prototype of BurstFS with log-structured write support and evaluated its performance and scalability. Fig. 2(a) compares the bandwidth of BurstFS with CRUISE by having two processes on the same node each write 1GB data to their local DRAMs following both N-1 and N-N patterns. BurstFS demonstrated 1.83× bandwidth improvement over CRUISE in N-1 pattern since the log-structured approach appends write requests one after another even when they are noncontiguous, while CRUISE will fill up the gap between two noncontiguous write requests with 0s in order to maintain the layout of file. Fig. 2(b) shows the scalability of BurstFS. 16 processes were placed on each node. All processes wrote to their local SSDs. The bandwidth of BurstFS scaled linearly with increasing process count.



(a) Bandwidth of BurstFS with Different Write Patterns.



(b) Scalability of BurstFS.

Fig. 2: Performance and Scalability of BurstFS.

We built the metadata service on top of MDHIM [5], a distributed key-value store for HPC systems. While leveraging its existing APIs, we extended its functionality to better support BurstFS semantics. For instance, retrieving all metadata of a contiguous file extent may require a range query across multiple nodes; we extended the support for these semantics. We also optimized the bulk put and get operations. Fig. 3 compares the performance of bulk put and get operations in BurstFS with the original MDHIM. In a bulk put operation, each client puts 1 million 40B key-value pairs in the metadata servers. Each key-value pair contains the information of a file segment, such as the file ID, offset in the file, etc. In a bulk get operation, each client got 1 million key-value pairs pointing to segments on a contiguous extent of a shared file. The total number of clients was fixed to 128. One client was placed on each node, and servers were collocated with clients as separate threads. The performance of both the original MDHIM and BurstFS scaled with increasing server counts. MDHIM performed a seek operation for each put in order to check if there was an old key-value pair, and combined the value of old one with the new one if needed. Since BurstFS only needed to replace the old key-value pair, it avoided this overhead and reduced the overall time by 27% on average. In addition, by replacing repetitive seek operations with one sequential scan for bulk get, BurstFS reduced the overall time by 91% on average.

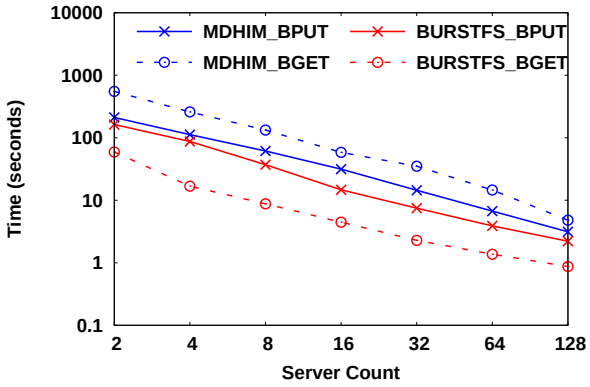


Fig. 3: Performance of Metadata Service

IV. Conclusion and Future Work

In this study, we proposed BurstFS, a distributed BB file system for node-local BB. Our implementation increased bandwidths by 1.83× for N-1 workloads and the initial evaluation demonstrates it is a promising storage solution for the next-generation leadership computing facilities.

In the future, we plan to finalize the implementation of distributed metadata service and RDMA-based data read, and evaluate BurstFS against real-world I/O workloads.

Acknowledgments

This work was supported in part by a research grant from Lawrence Livermore National Lab to Florida State University. This work was also under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. (LLNL-ABS-675972).

References

[1] Catalyst. http://computation.llnl.gov/computers/catalyst.
[2] Hyperion. https://hyperionproject.llnl.gov/index.php.
[3] Summit. https://www.olcf.ornl.gov/summit/.
[4] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate. PLFS: A Checkpoint Filesystem for Parallel Applications. In *SC*. ACM, 2009.
[5] H. N. Greenberg, J. Bent, and G. Grider. MDHIM: a parallel key/value framework for HPC. In *HotStorage*. USENIX Association, 2015.
[6] R. Rajachandrasekar, A. Moody, K. Mohror, and D. K. Panda. A 1 PB/s File System to Checkpoint Three Million MPI Tasks. In *HPDC*. ACM, 2013.
[7] K. Sato, K. Mohror, A. Moody, T. Gamblin, B. R. De Supinski, N. Maruyama, and S. Matsuoka. A User-Level Infiniband-Based File System and Checkpoint Strategy for Burst Buffers. In *CCGRID*. IEEE, 2014.
[8] T. Wang, S. Oral, Y. Wang, B. Settlemyer, S. Atchley, and W. Yu. Burstmem: A High-Performance Burst Buffer System for Scientific Applications. In *BigData*. IEEE, 2014.
[9] T. Wang, O. Sarp, M. Pritchard, B. Wang, and W. Yu. TRIO: Burst Buffer Based I/O Orchestration. In *CLUSTER*. IEEE, 2015.
[10] Y. Wang, R. Goldstone, W. Yu, and T. Wang. Characterization and Optimization of Memory-Resident MapReduce on HPC Systems. In *IPDPS*. IEEE, 2014.