LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-SR-742423

Final Report for File System Support for Burst Buffers on HPC Systems

W. Yu, K. Mohror

November 28, 2017

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Final Report
for
File System Support for Burst Buffers on HPC Systems

Weikuan Yu (yuw@cs.fsu.edu, 850-644-5442)

1 Summary

Distributed burst buffers are a promising storage architecture for handling I/O workloads for exascale computing. As they are being deployed on more supercomputers, a file system that efficiently manages these burst buffers for fast I/O operations carries great consequence. Over the past year, FSU team has undertaken several efforts to design, prototype and evaluate distributed file systems for burst buffers on HPC systems. These include MetaKV: a Key-Value Store for Metadata Management of Distributed Burst Buffers, a user-level file system with multiple backends, and a specialized file system for large datasets of deep neural networks. Our progress for these respective efforts are elaborated further in this report.

2 MetaKV: A Key-Value Store for Metadata Management of Distributed Burst Buffers

Distributed burst buffers are a promising storage architecture for handling I/O workloads for exascale computing. Their aggregate storage bandwidth grows linearly with system node count. However, although scientific applications can achieve scalable write bandwidth by having each process write to its node-local burst buffer, metadata challenges remain formidable, especially for files shared across many processes. This is due to the need to track and organize file segments across the distributed burst buffers in a global index. Because this global index can be accessed concurrently by thousands or more processes in a scientific application, the scalability of metadata management is a severe performance-limiting factor. In this paper, we propose MetaKV: a key-value store that provides fast and scalable metadata management for HPC metadata workloads on distributed burst buffers. MetaKV complements the functionality of an existing key-value store with specialized metadata services that efficiently handle bursty and concurrent metadata workloads: compressed storage management, supervised block clustering, and log-ring based collective message reduction. Our experiments demonstrate that MetaKV outperforms the state-of-the-art key-value stores by a significant margin. It improves put and get metadata operations by as much as 2.66× and 6.29×, respectively, and the benefits of MetaKV increase with increasing metadata workload demand.

These results have been published in the 2017 IPDPS conference as a full research paper.

3 A User-Level File System with Multiple Backends

An efficient file system is important for high-performance computing (HPC) systems in supporting large-scale scientific applications. In general, kernel-level file systems are generalized for broader Unix-like environments, while user-level file systems are designed with special-purpose features for various I/O workloads. The Filesystem in Userspace (FUSE) [3] is a software framework for Unix-like file systems, which allows non-privileged users to create their file systems without kernel-based file system implementations. Since most user-level file systems are designed to support diverse I/O workloads, different file systems can be used for different kinds of data in a single job. However, interacting with multiple FUSE file systems is a complicated task, due to the communication round trip between an application program and file system processes, and the need of root privilege to mount file systems.

We have designed Direct-FUSE as a framework that supports user-level file systems without crossing the kernel boundary. Direct-FUSE is built on top of libsysio, which is developed by Sandia Scalable I/O team and provides a POSIX-like interface which redirects I/O function calls to particular file systems. Direct-FUSE is designed to support various user-level file systems in one job with a unified POSIX-like interface, numerous backends, and less overheads. In this work, we make the following contributions: 1) We evaluate the overheads introduced by the round trips in FUSE file system calls and give a detailed cost breakdown analysis. 2) We describe the design of Direct-FUSE, which facilitates the capability of using multiple user-level file systems as backends for different I/O workloads. 3) We evaluate Direct-FUSE

performance with other FUSE-based local or distributed file systems. We also carry out a cost analysis for Direct-FUSE and compare it with native file systems and FUSE file systems.

These results have been included in the 2017 PDSW workshop for presentation as a WIP poster.

4 A Specialized File System for Large Datasets of Deep Neural Network

Recent deep learning has been largely driven by the ability of training large models on vast amounts of data. High Performance Computing (HPC) seems like playing an increasingly important role in helping deep learning to handle orders of magnitude larger training data for neural networks, because of the computing capability of current and upcoming HPC systems. As larger datasets lead to higher accuracy during the training, a dataset may not be fully cached in parallel file systems. Moreover, not all HPC systems enable local storage for every compute node. Therefore, to fully facilitate the current HPC utilities without additional cost on purchasing new equipment for new deep learning tasks, we make most of memory on every compute node to store dataset and allow the memory to be exposed to RDMA (Remote Direct Memory Access) for remote read.

Our DNNFS is composed by delegated servers and clients (application processes). As a server, it has data buffer, receive buffer, message buffers, and also an I/O manager. The I/O manager on each node, distributes datasets across nodes, builds RDMA connection during initialization, and addresses local or remote read requests. The data buffer is preserved as shared memory that is exposed to RDMA for allowing fast one-sided read for images. Since all images in training have the same resolution, images on are easily fetched through image ID. The receive buffer is also a shared memory exposed to RDMA for locally or remotely fetched data, and severing the data request from applications. As every read data segment has the same size, we manually divide the receive buffer into several constant size slots. Therefore, images can be directly fetched from the receive buffer by clients based on slot ID. Message buffers are for exchanging data buffer's MR (Memory Region), sending and receiving connection and disconnection messages. For DNNFS client (application process), it communicates with the delegated server via file pipe and fetches data from shared memory. The file pipe is for small messages transformation between clients and servers, which includes the notification of data request and the available slot ID for client to fetch image from the receive buffer.

In our experiments, we evaluated uploading speed and the read bandwidth on client side. We measure the uploading bandwidth with different number of nodes and different transfer sizes. ImageNet32 is used as test dataset, whose images are all resized to 32 * 32. The aggregate uploading bandwidth is linear with the compute nodes count, which varies from 2 GB/s to around 29 GB/s scaling from 1 node to 16 nodes. The different transfer sizes tests are run on 16 nodes. The transfer speed is relatively stable when the transfer size is greater than 64 KB, which is around 29 GB/s. In the client image read bandwidth experiment, we upload random number on memory of compute nodes to emulate pixel values of images. During the test, we continually issued 1000 read requests on a client side with resolution 32 * 32 (3 KB), 64 * 64 (12KB), 256 * 256 (192 KB), 512 * 512 (768 KB). Based on the current experimental results, the bandwidths are around 0.25 GB/s, 0.7 GB/s, 1.85 GB/s, and 2.17 GB/s respectively.

This is an ongoing project, we will continue working on incorporating our design with TensorFlow, analyze how much performance improvement. We also plan to propose new optimization methods for large HPC system to further enhance performance.