Utilizing In-Memory Storage for MPI-IO

Julian Kunkel
Deutsches Klimarechenzentrum GmbH
Bundestraße 45a
20146 Hamburg
Email: kunkel@dkrz.de

Eugen Betke
Deutsches Klimarechenzentrum GmbH
Bundestraße 45a
20146 Hamburg
Email: betke@dkrz.de

***Abstract*—In contrast to disk or flash based storage solutions, throughput and latency of in-memory storage promises to be close to the best performance. Kove®'s XPD® offers pooled memory for cluster systems. However, the system does not expose access methods to treat the memory like a traditional parallel file system that offers POSIX or MPI-IO semantics.**

Contributions of this poster are: 1) Implementation of a MPI-IO driver for the XPD. 2) Thorough performance evaluation of the XPD using IOR with MPI-IO mode. This MPI independent file driver enables high-level libraries (HDF5, NetCDF) to utilize the XPD's pooled memory. We demonstrate that the MPI-IO driver is able to efficiently take benefit of the pooled memory by providing it as in-memory storage.

I. Introduction

The dilemma of the conventional high-performance storage systems is that they must maximize the bandwidth to reduce application runtimes and at the same time they shall minimize the available bandwidth to reduce costs. The first requirement is often priorized to the detriment of the second one, which typically ends up in the oversizing and in a low average usage of the expensive bandwidth. The priorization is motivated by the large performance peaks, that often occur in large-scale applications [1].

In an alternative storage architecture, a burst buffer is placed between compute nodes and the storage and acts as an intermediate storage tier. The goal is to catch the I/O peaks from the compute nodes. Therefore, it provides a low latency and good bandwidth to the compute nodes, but also utilizes the backend storage more efficiently by streaming data constantly at a lower bandwidth. The current flash-based storage systems, e.g., DDN IME [2], IBM FlashSystem [3], are fast enough to act as a burst buffer, but pooled memory, like the Kove® XPD® [4], provides better latency, endurance and availability and, therefore, may be even more suitable for this kind of job.

Since the XPD does not offer a method for coordinated parallel I/O, our objective is to provide MPI-IO [5]. MPI-IO is a widely accepted middleware layer for parallel I/O. It allows the large-scale applications to share files efficiently, facilitating the programming efforts and is a part of many libraries and used by a wide range of applications. Many of the current MPI-IO implementation are optimized for the conventional storage and their I/O behaviour is well understood, but the new technology still requires a thorough analysis.

II. XPD KDSA API

The XPD KDSA API is a low-level API that allows to send and receive data using write/read calls by utilizing RDMA. Data can be transferred synchronously or asynchronously, additionally, memory can be pre-registered for use with the Infiniband HBA or unregistered transfers can be initiated. Since registration of memory is time consuming, for unregistered memory regions the system may either use an internal (pre-registered) buffer and copy the user's data to the buffer, or for larger accesses it registers the memory, performs an RDMA data transfer and then unregisters the memory again.

To address a XPD volume as a virtual address space, the XPD uses a connection specifier in the form <local_address>/<server>.<link>:<volume ID>. Multiple volumes and client or server links can be aggregated by adding them with a +, data is then striped across these volumes using the specified links. Similar to parallel file systems, this allows to scale the number of connections with the requirements.

III. XPD-MPIIO-driver

The driver is implemented as a shared library and usable with any MPI. It can be selected at startup of an application using LD_PRELOAD with the shared library. All implemented routines check the file name for the prefix "xpd:". If they are not finding it, they route the accesses to the underlying MPI. Therewith, some files can be stored on any storage supported by the basic MPI but also some others at different XPD volumes. The source code is available at http://github.com/JulianKunkel/XPD-MPIIO-driver.

The file driver implements important functions utilizing the relaxed consistency semantics offered by MPI-IO: MPI_File_open(), close, delete, get_position, get_size, preallocate, read_at, write_at, read_at_all, write_at_all, read, write, seek, set_size, set_view and sync. Collective read/write are calling the independent counter part. The selection is inspired by the needs of HDF5 and IOR. The implementation comes with a few limitations. Since we do not know the memory regions, the KDSA calls for unregistered memory are used implying overhead as described above. Views do not support derived data types (still HDF5 works). During the open/close the Infiniband connections to the XPD's are established and destroyed. This implies an additional overhead comparable to parallel file systems but offers the freedom to choose the volumes on a file basis.

Internally, the file driver uses the space provided by one or multiple volumes. It records the actual file size at the beginning of this memory region but cannot grow beyond the aggregated size of the volumes. Each process tracks its view of the file size and exchanges this information upon file close or flush as needed by MPI-IO semantics. The data space is not initialized with zeros, which is an issue if files are written in a sparse format. A formatting tool is contained in the repository that initializes file size or completely initializes memory regions.

IV. Performance Evaluation

Our goal is to systematically investigate the scaling behavior of the Kove XPD's. As a benchmark, IOR is used varying access granularity, processes-per-node, nodes, XPD connections and access pattern (random and sequential). IOR is used with a transfer size equal to the access granularity and 20 GiB of data per XPD connection (and volume)[1]. To synchronize the measurements and capture time for open, close and I/O separately, inter-phase barriers are turned on (IOR option -g).

The tests were run on Cooley, the visualization cluster of Mira on ALCF. It provided three XPD's with a total of 14 FDR connections. Each node is equipped with one FDR HBA.

Since the storage capacity is rather low compared to the speed of the tests, the time for open/close are investigated separately. In average, the time for open/close reduces the reported performance by 10%. However, for production runs, larger capacities are assumed, reducing this overhead. Therefore, the reported performance is computed without the open/close time.

The following investigations are made: 1) scaling on 14 nodes with increasing number of connections, 2) scaling for 14 connections, 3) variability of performance, 4) time for open/close.To estimate the behavior for larger systems, a linear model is build and applied for systems with 10,000 nodes. Finally, a comparison to a decent Lustre System of DKRZ's supercomputer Mistral[2] is made.

For the Lustre benchmarks we were trying to reuse the XPD parameters wherever possible. Collective buffer was enabled for write operations smaller than 512 KiB, we configured MPI-IO to use one aggregator per node and, in all cases the number of stripes was twice as much as the number of nodes.

V. Observations & Conclusion

The observations from the evaluation are summarized as follows: 1) read/write behave mostly symmetrically, i.e., good read performance implies good write performance, 2) sequential and random behaves similarly (not shown in the poster), 3) performance scales well with the number of client nodes and the provided server connections, 4) for small access granularities, the workload is dominated by the latency implied by Infiniband and the compute overhead, thus, it improves beyond 14 client nodes and using more PPN. 5) for large access granularities, a high percentage of peak is achieved quickly. 6) performance of 100 KByte accesses is higher than for 1 MiB in many cases, this is due to the pre-registered memory region inside the KDSA library that is used for small accesses but not for 1 MiB. Therefore, the overhead for memory registration is added which slows down the I/O.

7) The variability of access time has been investigated, showing that repeated runs exhibit a very similar performance behavior. In our benchmark the mean values were 1.23% for read and 1.78% for write access, which are uncommonly low for parallel file systems.

8) Time for opening/closing of a shared file is for parallel file systems sometimes high, therefore, this is investigated for the MPI wrapper. Each client process establishes its own connections to all XPDs specified in the connection string, thus, the performance is expected to depend on the number of nodes, PPN, and connections. Up to the 100 nodes, the time is below 2s and 0.5s for open/close, respectively. With the help of linear models, the time for scaling to large systems could be roughly estimated; it is 30 minutes for 10,000 nodes each talking to all XPDs. Another setup would be to connect clients to only a small subset, e.g., to use it as write buffer or for stencil type I/O. With 14 connections per client, the opening takes 26s. Initially, open/close took 30 times longer, but Kove optimized the behavior after we reported that this issue is relevant for this prototype.

XPDs significantly outperforms our Lustre system in the small-blocks random I/O benchmarks. In this case and in contrast to XPD, the increasing number of nodes and processes don't provided the desired scaling effect. The performance benefit of the XPD is smaller when we use large granularities.

From these results, it appears that the XPD and this wrapper can be used to support I/O heavy workloads. In the future, we will extend the prototype to support additional MPI semantics such as file views, evaluate real application workloads and provide asynchronous flushes for burst-buffer scenarios.

[1]The storage capacity of the XPD's is shared amongst all users, therefore, I had to deal with 20 GiB and 14 volumes.

[2]http://www.vi4io.org/hpsl/2016/de/dkrz/lustre02

Acknowledgment

This research is supported by Bull, atos technologies. Thanks to Kove for their support and discussion. Thanks to our sponsor William E. Allcock for providing access to Cooley and the feedback. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

[1] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *In Proceedings of the 2012 IEEE Conference on Massive Data Storage*, 2012.

[2] "Worlds's most advanced application aware I/O acceleration solutions," http://www.ddn.com/products/infinite-memory-engine-ime14k, DDN, accessed: 2016-07-12.

[3] "Flash Storage," http://www-03.ibm.com/systems/storage/flash, IBM, accessed: 2016-07-12.

[4] *about xpress disk (xpd)*, Kove Corporation, 2015.

[5] R. Thakur, W. Gropp, and E. Lusk, "On Implementing MPI-IO Portably and with High Performance," in *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, 1999, pp. 23–32.