ORNL/TM-2016/415

Impact of Burst Buffer Architectures on Application Portability

Kevin Harms (ANL)
Sarp Oral
Scott Atchley
Sudharshan Vazhkudai

September 30, 2016

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2016/415

National Center for Computational Science

Impact of Burst Buffer Architectures on Application Portability

Kevin Harms (ANL)
Sarp Oral
Scott Atchley
Sudharshan Vazhkudai

Date Published: September 30, 2016

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-BATTELLE, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

Table of Contents

ABSTRACT

The Oak Ridge and Argonne Leadership Computing Facilities are both receiving new systems under the Collaboration of Oak Ridge, Argonne, and Livermore (CORAL) program. Because they are both part of the INCITE program, applications need to be portable between these two facilities. However, the Summit and Aurora systems will be vastly different architectures, including their I/O subsystems. While both systems will have POSIX-compliant parallel file systems, their Burst Buffer technologies will be different. This difference may pose challenges to application portability between facilities. Application developers need to pay attention to specific burst buffer implementations to maximize code portability.

1. INTRODUCTION

The two U.S. Department of Energy (DOE) leadership computing facilities, the Oak Ridge Leadership Facility (OLCF) at Oak Ridge National Laboratory (ORNL) and the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory (ANL), provide world-leading computing capabilities to scientists and engineers for accelerating major scientific discoveries and engineering breakthroughs for humanity, in partnership with the computational science community. Both OLCF and ALCF have deployed different HPC architectures that are 10 to 100 times more powerful than typical research computing systems.

Access to OLCF and ALCF platforms is provided under the The Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program [1]. The INCITE program is operated by the OLCF and ALCF, and research projects are awarded machine allocations on the OLCF and ALCF supercomputers. The selected projects address grand challenges in science and engineering. The projects select either one or both systems to apply for time on, typically based on which computer architecture best matches their software.

Under the CORAL collaboration, OLCF and ALCF will deploy two new pre-exascale systems by 2018 [2]. OLCF's system, Summit, will be built by IBM and ALCF's system, Aurora, will be built by Intel and Cray. These two systems will represent two different computer architectures and two vastly different approaches for pre-exascale systems. The Summit system features a smaller number (~3,500) of "fat" nodes, which will have a heterogeneous architecture combining IBM's POWER CPUs and NVIDIA GPUs, connected via a traditional fat-tree, InfiniBand fabric. On the other hand, the Aurora system will feature a large number (~50,000) of "thin" nodes, which will have a homogeneous architecture using Intel's third generation Xeon Phi processors, connected by a new Intel interconnect with a dragonfly topology.

Summit and Aurora will both have large capacity and POSIX-compliant parallel file systems (PFS). The Summit PFS will be built around IBM's GPFS technology [3], while Aurora will have Intel's Lustre technology [4]. In addition to the PFS, both Summit and Aurora will have Burst Buffer (BB) systems [5, 6] to augment the PFS. The CORAL requirements had high PFS capacity and bandwidth requirements. However, with current technology it is not cost-effective to realize these requirements in a single PFS layer built only using traditional magnetic spinning disks. Therefore, both of these systems are designed with under-provisioned PFS in terms of bandwidth and capacity. The newly introduced BB layer on these systems is expected to alleviate the under-provisioning of the deployed PFS, due to bursty I/O that is common in HPC due to tightly-coupled, bulk synchronous programming. The BB is specified to accept the bursty I/O and then drain them to the underlying PFS at a lower rate. The expectation is that the BB services will reduce the I/O bandwidth requirements from the PFS and will compensate for the lack of bandwidth out of the PFS. Summit and Aurora will have fundamentally different BB architectures.

Aurora will feature a centralized BB that appears as a mounted file system on the compute nodes. Summit will have a BB that is based on node-local persistent storage.

Since applications can request time on either Summit or Aurora, it is critical that they be able to run on either of these machines. However, the fundamentally different compute node and BB architectures pose numerous challenges when it comes to application portability. As scientists begin to think about application portability across these systems, it is natural to only focus on the processor differences and how to achieve FLOPS on these systems, while I/O is a secondary (or tertiary) consideration. The different processor architectures are likely to throw numerous challenges given the multiple CPUs/GPUs versus many-core models. Be that as it may, the goal of this paper is to raise awareness on the I/O challenges that application programmers are likely to face in porting codes across Summit and Aurora. The disparate burst buffer architectures on these systems suggests that I/O cannot be an afterthought, and needs to be carefully considered with several design choices. Thus, in this paper, we highlight the different BB use cases, the BB architectures and their pros and cons, and some best practices to adopt when porting codes.

2. BURST BUFFER USE CASES

The CORAL requirements stated that, at a minimum, the BB system needs to support rapid checkpoint/restart to reduce the I/O node-to-file system performance requirements by an order of magnitude [2]. In addition to this requirement, it was expected that the BB will be integrated into the end-to-end I/O solution. Possible envisioned CORAL BB use cases are listed below.

- *Checkpoint/Restart* BB will be used as a means to store checkpoint data, in order to provide a fast, reliable, performance impedance matching storage space for applications. BB will drain the checkpoint data to the PFS, while also supporting the ability to restart applications from the checkpoint data stored therein. Applications use different I/O models including file per job (N:1), file per process (N:N), or a mix of these two (N:M) and, ideally, the BB would support all.
- *Stage-in and Stage-out* BB may be used as a staging ground to bring an application's input data closer to a job. Similarly, the result output data of an application may be staged out to the BB, before being migrated to its final destination.
- *Data Sharing* BB may be used as a conduit to enable data sharing between consecutive jobs running on the same machine. Some architectures could enable sharing between jobs on different machines in the center. Thus, BB may be used to tie together the components of an end-to-end simulation workflow, and it may be used to start a subsequent job from the most recent checkpoint of the preceding job.
- *Write-through Cache in the File System* BB may be used as a write-through cache within the file system storage targets to expedite regular I/O and not just checkpoint data.
- *In-situ Analysis* BB may be used to facilitate in-situ analysis of the checkpoint snapshot data or the result data. In-situ analysis is when a concurrent job that runs alongside the simulation job, whose output (reduced) may also be written to the burst buffer.

The priorities of these use cases are directly related to facilities' computational workflows. For OLCF while the checkpoint/restart and write-through cache in the file system use cases are identified as high priorities, in-situ analysis is marked as a medium priority and data sharing and stage-in and stage-out use cases are declared as low priorities. ALCF has the same high priorities as OLCF with the exception that stage-in/out would be more medium priority.

3. BURST BUFFER ARCHITECTURES

There are three different BB architectures possible, including compute node-local, co-located with the I/O nodes, and a separate set of nodes. We describe each design and address their advantages and disadvantages below.

3.1 COMPUTE NODE-LOCAL STORAGE

In this design, the storage devices are located in the compute nodes. The node's OS creates a local POSIX file system on the device and application I/O uses the normal storage stack. In order to enable asynchronous draining of data to the PFS, the node needs to have a dedicated BB thread or daemon and the BB API should provide a call to initiate a background transfer. The user's access to the BB is limited to the job run, and optionally a small amount of time before and/or after. The user must migrate data out of the BB during or shortly after the job completes.

3.1.1 Advantages

The advantages of this design include: exclusive and fair access to the storage devices, linear scaling of BB performance with the number of compute nodes, no network traffic during application use of the BB, extended memory, and lower cost. When reading or writing, the node's processes have exclusive use of the storage resource, which provides more consistent and predictable performance. The various nodes within the job are guaranteed fair access, because there is no sharing. The I/O performance scales by the number of nodes times the device performance so that larger jobs get more BB performance. During application I/O, there is no network traffic and the bursty nature does not impact any shared resources (i.e. network or PFS). With a node-local device, the application can use the device to extend memory, which could allow for larger working sets. Lastly, the cost is limited to adding a storage device to the compute node.

3.1.2 Disadvantages

The disadvantages include: difficulty supporting N:1 model, the need for a background thread/daemon, stage-in and stage-out require network traffic to the compute nodes, lack of data resiliency, and maintenance of the storage device requires taking the node offline. Without a shared namespace (i.e. mount point), this architecture cannot support the N:1 mode nor can it support sharing data between nodes (i.e. one node reading another node's output). In order to do so, it would require the user to instantiate a PFS such that the compute nodes serve as I/O nodes and one or more compute nodes serve as the metadata servers, which would add to jitter (i.e. increase runtime variability) as well as consume local resources such as memory and compute cycles. An alternate design could use a set of I/O nodes or other non-compute nodes to host the metadata servers, but each compute node would still need a I/O server. Even without support for N:1, this design requires each node to have a background thread or daemon to enable asynchronous draining to the PFS, although given the amount compute capabilities on these systems this might not be too much of a burden. If the system provides stage-in and stage-out services, these services will impose additional network traffic on the compute nodes. If these services allow overlapping use between jobs (i.e. job N starts while job N-1 is draining), this will lead to shared use of the local resource thus increasing jitter. The BB is comprised of local devices and provides no data resiliency should a node fail. In order to provide data resiliency, applications will need to use libraries, such as SCR [7]. Lastly, any maintenance required by the storage device may require taking the entire node offline, which negatively impacts facility utilization. Typically, BB systems employ non-volatile memory (NVM) because of its higher performance, but NVM wears out over time, which raises a maintenance concern.

3.1.3 Use Case Support

Of the five use cases outlined in Section 2, this design can satisfy three of the five use cases: checkpoint/restart, staging, and in-situ analysis. Support for some of these use cases, however, comes with severe constraints. The staging use case is problematic. If the resource manager is staging data to the BB for the next job and one of the allocated compute nodes fails, the resource manager will need select another node and start staging over for that node, which could possibly hold up the job start. Another issue for staging is that the user has to somehow provide information to the resource manage on how to map files or portions of files to the various compute nodes. The in-situ analysis use case is restricted to the analysis job running on the same nodes as the simulation job. This might be reasonable if the simulation cannot use all of the resources of the node and if the analysis can make use of node-local only data.

3.2 CO-LOCATED WITH I/O NODES

This design places the BB functionality on the I/O nodes. It could be implemented as a fast resource pool within the same namespace or as a separate namespace (i.e. a different mount point). The compute nodes directly mount the BB namespace.

3.2.1 Advantages

The advantages include: support for N:1, lower jitter than node-local, low cost, data resiliency, longer residency times, and possibly the easiest ability to support stage-in and stage-out. With the shared namespace, this design can support both N:N and N:1 models including sharing data between nodes. This design does not require a background thread or daemon, but it does rely on the OS to push data to the BB. The cost is limited to the number of devices added to the existing I/O node hardware. The cost could be lower than the node-local design if the number of storage devices is less than the number of compute nodes. Because the PFS typically has support for data resiliency, this design does not need library support to provide resiliency. Depending on the capacity of the BB, it could allow longer residency times before mandatory draining to the PFS. Support for staging between the PFS and BB, depending on the design, might be the easiest to implement and would not add any traffic to the network.

3.2.2 Disadvantages

The disadvantages include: shared use of the BB between jobs, negatively impacts PFS performance, and the loss of the PFS would mean the loss of the BB as well. Because all jobs would share access to the BB, performance would be variable and difficult to predict. Being co-located with the PFS I/O nodes, the BB and the PFS would share network connections meaning that the bursty BB traffic would negatively impact PFS performance. And of the most concern, if the PFS goes offline, then so does the BB.

3.2.3 Use Case Support

This design supports all five of the use cases and with the least restrictions.

3.3 SEPARATE SET OF NODES

This design places the BB on a dedicated set of nodes separate from the compute nodes and the I/O nodes. The BB can be a separate namespace from the PFS or integrated within the PFS. The compute nodes can mount the BB directly or use a function-shipping library.

3.3.1 Advantages

The advantages include: support for N:1, lower jitter than node-local, no negative impact on PFS, somewhat longer residency times, and possibility of continued use if the PFS goes offline. As with the co-located design, this design supports both N:N and N:1 models. It would not require a background thread/daemon, but it too would rely on the OS to move the data. Because the nodes are separate from the I/O nodes, the bursty BB traffic would not impact the PFS directly. The additional traffic on the network might add to congestion, which could negatively impact the PFS. This design may allow longer residency times than node-local designs, but the additional time is limited by the capacity of the BB. Lastly, depending on the implementation, the BB system could remain operational if the PFS goes offline.

3.3.2 Disadvantages

The disadvantages include: shared use of the BB between jobs and the highest cost of all the designs. As with the co-located architecture, this design features a shared use model, which increases performance variability and reduces predictability. The dedicated nodes add much more cost both in terms of additional servers and in additional network hardware including NICs and possibly switches.

3.3.3 Use Case Support

This design supports all five of the use cases, albeit with one restriction. The write-through cache requires integration between the BB and the PFS. The issue for the compute node is where to find the data, which is a coherency issue.

4. APPLICATION PORTABILITY

The I/O architecture of both leadership systems require that the applications must take advantage of the burst buffer whenever possible. In the CORAL request for proposal, the specification was that the burst buffer would have 10 times the bandwidth of the PFS. It is imperative that users take full advantage of the BB services. Users, who bypass the BB services and directly access the PFS on these systems, will not only hurt themselves, but will also lower the overall PFS performance for all users on the system. Therefore, it is critical for users to adapt their codes to take advantage of the BB and also to have portable codes between OLCF and ALCF systems in terms of BB services.

The burst buffer architectures for these two systems offer different capabilities with regard to how an application does I/O. Currently, there is no standard interface or behaviors for BB service. This means that application developers will need to handle differences in the API and semantics in the I/O model for the burst buffer. Research in the areas around burst buffers and multi-level storage hierarchies are ongoing [8]. Representatives from the CORAL and TRINITY/NERSC-8 consortium (Los Alamos, Sandia, and Lawrence Berkeley National Laboratories) [9] are also working towards a standardization of the BB API to represent the BB architectures of the corresponding procurements (Summit, Aurora, Cori, and Trinity machines). However, it is unlikely that any standardization will arrive in the Aurora and Summit time frames.

Currently, the publicly available information about next generation burst buffer software is limited, however some possible best practices can be discussed. The N:1 model will likely be difficult to support on a strictly node-local storage architecture, such as Summit, therefore a N:N model should be portable between Summit and Aurora architecture.

The N:N model may also be adapted to support a limited N:M model, where MPI ranks local to the same node may share storage objects if the application can effectively determine which ranks share a node. A fully generalized N:M model may allow applications to use centralized BB architectures more efficiently in cases where a full N:N model may generate too many storage objects.

The standard POSIX file API will likely be supported by all burst buffer services, however it is unclear if POSIX semantics will be strictly enforced or will have the highest performance. Applications will likely be more portable if they do not rely on implicit locking or resolution of writing overlapping I/O regions.

The final aspect of the BB service is managing data transfers and any stage-in (moving data objects from the PFS to the BB prior to application execution) or stage-out support (moving of data objects from the BB to the PFS during or post application execution). These services may be controlled by a platform specific BB API and schedulers. The BB API from the different vendors will likely offer similar functionality, but with incompatible APIs. Applications will be more successful if they can isolate this portion of code within their application. Further, integration with a check-point library, e.g., SCR, could perhaps obviate the need for an application to explicitly integrate with platform-specific BB APIs. For example, on Summit, the intent is for SCR to be integrated with IBM's BB stage-out APIs, and as long as the application uses SCR, stage-out could happen seamlessly. Else, the application is responsible for triggering the drain of the checkpoint from the BB to the PFS. Therefore, even though the POSIX mount point of the compute node-local BB provides a convenient interface that applications can write to without any instrumentation, the fact that the data needs to be drained to create room for future checkpoints means that it can be beneficial for applications to integrate with a checkpoint library that can manage the draining. This will be true on Aurora too, as the data from the central BB needs to be drained to the PFS, although we can afford to leave it on the BB a little longer compared to the node-local BB, as long as the shared BB storage system's purge policy permits it.

Table 1: Example I/O checkpoint which shows the portable portion of code as well as BB code that would

```
1 int write checkpoint (int rank , const void *data , size t data len)
2 {
3     void *set;
4     int fd;
5     char filename[MAX_PATH];
6
7     /* /mnt/bbfs would be site specific mount point for accessing BB
8        service checkpoint <rank>.dat is a unique name that will work for
9        cases where BB is centralized or node-local */
10    sprintf(filename , "/mnt/bbfs/checkpoint %d.dat", rank);
11
12    BBAPI create set(&set ); /* Internal tracking for BB service */
13

14    /* Portable POSIX code */
15    fd = open(filename , OCREAT+OTRUNC+OWRONLY);
16    write(fd, data, data len);
17    close(f);
18
19    BBAPI add file(filename , &set ); /* file that BB service will transfer */
20    BBAPI start transfer(&set ); /* tells BB to start transfer in */
21    BBAPI free set(&set ); /* background */
22 }
```

4.1 CODE SAMPLE

Table 1 has a simple code example of what a portable I/O solution will look like on the CORAL platforms. Lines 15 through 17 show the chunk of POSIX I/O code which will be fully portable. Line 10 highlights the need for appropriate file naming with regard to different possible burst buffer implementations. The different process-centric, node-centric or centralized BB service may have differences in handling file name solution, however keeping unique file names between all processes will work under any implementation. Lines 12 and 19-21 demonstrate interaction with the platform specific BB API. This example uses a fictitious BB API but illustrates the type of API calls that may need to be made.

5. CONCLUSIONS

The CORAL procurement will bring two architecturally diverse, high performance computing architectures to the DOE Office of Science user facilities. Application portability based on the computing architecture is only one consideration for overall portability. The I/O architecture will also be an important consideration and should not be overlooked. The application solution should not ignore the BB services in favor of the portability offered by the parallel file system, because the potential performance loss will be far too great. Bypassing the BB will not only negatively impact the user, but all users of the facility.

Acknowledgements

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.

This research also used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

6. BIBLIOGRAPHY

[1] U.S. Department of Energy, "INCITE Program," [Online]. Available: http://www.doeleadershipcomputing.org/.

[2] CORAL Collaboration - Oak Ridge, Argonne, Livermore, "CORAL Request for Proposal B604142," January 2014. [Online]. Available: https://asc.llnl.gov/CORAL.

[3] F. B. Schmuck and R. L. Haskin, "Gpfs: A shared-disk file system for large computing clusters," in *FAST*, 2002.

[4] P. J. Braam et al, "The lustre storage architecture," 2004.

[5] G. Grider, "Exa-scale fsio, can we get there? can we afford to?," in *HEC FSIO 2010 Workshop*, Arlington, VA, 2010.

[6] 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 *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, 2012.

[7] A. Moody, G. Bronevetsky, K. Mohror and B. de Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, New Orleans, 2010.

[8] DOE, "Storage systems and input/output to support extreme scale science," 2014 Dec. [Online]. Available: http://science.energy.gov/~/media/ascr/pdf/programdocuments/docs/ssio-report-2015.pdf.

[9] Trinity and NERSC-8 Alliance, "Trinity and NERSC-8 Computing Platforms: Draft Technical Requirements," 2013. [Online]. Available: http: //www.nersc.gov/users/computational-systems/ cori/nersc-8-procurement/trinity-nersc-8-rfp/.