Performance Characterization of Scientific Workflows for the Optimal Use of Burst Buffers

C.S. Daley[a,*], D. Ghoshal[a], G.K. Lockwood[a], S. Dosanjh[a], L. Ramakrishnan[a], N.J. Wright[a]

[a]*Lawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA 94720*

Abstract

Scientific discoveries are increasingly dependent upon the analysis of large volumes of data from observations and simulations of complex phenomena. Scientists compose the complex analyses as workflows and execute them on large-scale HPC systems. The workflow structures are in contrast with monolithic single simulations that have often been the primary use case on HPC systems. Simultaneously, new storage paradigms such as Burst Buffers are becoming available on HPC platforms. In this paper, we analyze the performance characteristics of a Burst Buffer and two representative scientific workflows with the aim of optimizing the usage of a Burst Buffer, extending our previous analyses [1]. Our key contributions are a). developing a performance analysis methodology pertinent to Burst Buffers, b). improving the use of a Burst Buffer in workflows with bandwidth-sensitive and metadata-sensitive I/O workloads, c). highlighting the key data management challenges when incorporating a Burst Buffer in the studied scientific workflows.

Keywords: Burst Buffer, DataWarp, Workflow, HPC

*Corresponding author

Email addresses: csdaley@lbl.gov (C.S. Daley), dghoshal@lbl.gov (D. Ghoshal), glock@lbl.gov (G.K. Lockwood), sudip@lbl.gov (S. Dosanjh), lramakrishnan@lbl.gov (L. Ramakrishnan), njwright@lbl.gov (N.J. Wright)

1. Introduction

The science drivers for high-performance computing (HPC) are broadening with the proliferation of high-resolution observational instruments and emergence of completely new data-intensive scientific domains. Scientific workflows that chain the processing and data are becoming critical to manage these on HPC systems. Thus, while providers of supercomputing resources must continue to support the extreme bandwidth requirements of traditional supercomputing applications, centers must now also deploy resources that are capable of supporting the requirements of these emerging data-intensive workflows. In sharp contrast to the highly coherent, sequential, large-transaction reads and writes that are characteristic of traditional HPC checkpoint-restart workloads [2], data-intensive workflows have been shown to often utilize non-sequential, metadata-intensive, and small-transaction reads and writes [3, 4]. However, parallel file systems in today's supercomputers have been optimized for more traditional HPC workloads [5]. The rapid growth in I/O demands coming from data-intensive workflows are demanding new performance and optimization requirements in the design of future HPC I/O subsystems [3]. It is essential to develop methods to quantitatively characterize the I/O needs of data-intensive workflows to ensure that resources can be deployed with the correct balance of performance characteristics.

The emergence of data-intensive workflows has coincided with the emergence of flash devices being integrated into the HPC I/O subsystem as a "Burst Buffer" (BB), a performance-optimized storage tier that resides between compute nodes and the high-capacity parallel file system (PFS). The BB was originally conceived for massive bandwidth requirements of checkpoint-restart workloads for extreme-scale simulations [6]. This is a workload characterized by distinct phases of application computation and I/O. The BB absorbs the I/O from the application which allows the application to resume computation as soon as possible. The data movement between the BB and PFS can then happen concurrently with application computation. However, the flash-based storage

media underlying BBs are also substantially faster than spinning disk for the non-sequential and small-transaction I/O workloads of data-intensive workflows. This motivates using BBs to accelerate diverse I/O workloads and enable use cases beyond buffering of I/O requests, such as providing a temporary scratch space, coupling workflow stages, and processing data in-transit [7].

BBs provide a unique opportunity to optimize I/O access patterns in scientific workflows executing on supercomputers. However, the design of a data management strategy using BBs requires careful consideration. The complex data access patterns and dependencies of data-intensive workflows can result in high data access penalties. Today, there is limited understanding of performance characteristics of BBs for data-intensive applications and the use of BBs with workflow systems. Thus, there is a need to understand how current and future workflow systems may use BBs effectively and efficiently when scheduling tasks, capturing provenance and providing fault tolerance.

In this paper, we analyze the I/O requirements of scientific workflows with the aim of being able to execute workflows more efficiently on supercomputers with BBs. We consider two scientific workflows that uses resources at the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory. Specifically, in this paper we make the following contributions:

- The development of a performance analysis methodology to analyze the performance characteristics of workflows pertinent to BBs.
- Demonstration of the usage of this methodology for the performance optimization of two scientific workflows with diverse characteristics running on a BB.
- Derived recommendations for future BB API capabilities to enable more straightforward matching of workflow performance requirements with requested resources for the two workflows.

Our paper provides a foundational framework in understanding the char-

acteristics of the BB and building workflow system support in the context of the scientific applications in a workflow. This paper extends our previous work [1] in the following ways: a). we include a performance characterization of a single unit of the BB in NERSC's Cori system using standard I/O benchmarks to obtain upper-bound measurements for I/O bandwidth, I/O operations per seconds (IOPS) and metadata rate, b). we analyze the I/O characteristics of workflow tasks to identify what will be the performance limiter as the workflow is scaled up, c). we show I/O scaling behavior for all workflow tasks and explain the loss of scaling by direct reference to the measured I/O characteristics of the workflow tasks and expectations about when the workflow tasks will saturate the storage resource, d). we demonstrate how a capability in the NERSC BB implementation can overcome metadata scaling bottlenecks in workflow tasks, and e). we draw attention to the key data management challenges which exist when trying to efficiently execute a workflow on a supercomputer with a BB. We should note that the focus of our work is about making optimal usage of the BB tier as opposed to improving scheduling decisions about when to move data between tiers, which is managed by workflow and batch queue systems. Nonetheless our detailed analysis of I/O characteristics allow us emphasize the capabilities required from BB software and middleware needed to move data and library files efficiently in an end-to-end workflow.

The paper is organized as follows. Section 2 presents an overview of BBs and gives details about the NERSC BB architecture. Section 3 details our approach to scalable I/O characterization for both workflows. Section 4 contains performance results relevant to the execution of the workflows on a BB. In Section 5 we discuss key findings related to efficient use of BBs as well as implications for the design of the next generation of BB storage architectures. We discuss related work in Section 6 and provide conclusions in Section 7.

2. Background

2.1. Burst Buffers

A BB is an additional tier in the storage hierarchy designed to accelerate application I/O. It is motivated by the high financial cost of deploying a single-tier storage system consisting of Hard Disk Drives (HDDs) only. It is expensive to use HDDs because there have only been minor performance improvements in the underlying magnetic disk technology over time. The implication of this trend is that supercomputer centers must add more HDDs to the storage system to deliver the improved I/O performance expected in a new supercomputer. An alternative to adding more HDDs is to use higher-performance Solid State Drives (SSDs) consisting of NAND-Flash memory. At the current time, NAND-Flash is the most widely produced Non Volatile Random Access Memory (NVRAM). The performance of enterprise SSDs containing NAND-Flash are $\sim 2-3$ GiB/s of I/O bandwidth and $\sim 10^4 - 10^5$ IOPS. This is more than an order of magnitude improvement in I/O bandwidth and $2-3$ orders of magnitude improvement in IOPS compared to HDDs. However, the current high cost of SSDs make it impractical to deploy a high-capacity single-tier SSD storage system. Therefore, an economically motivated storage system uses SSDs to meet the performance requirements and HDDs to meet the capacity requirements of the storage system. The SSD tier is used to create a BB.

BB Placement. Figure 1 is a high-level picture of a storage system in a supercomputing platform. The three main components are compute nodes, I/O nodes and storage servers. The compute nodes are connected to the I/O nodes by a High Speed Network (HSN), e.g. Cray Aries, and the I/O nodes are connected to the storage servers by a Storage Area Network (SAN), e.g. Infiniband. The I/O nodes therefore act as a bridge between the compute nodes and the storage servers for the application I/O traffic. The figure considers different placements of a BB, and specifically the NVRAM media, in a supercomputing platform. The first placement involves co-locating the NVRAM media with the compute nodes. This could be as a PCIe-attached SSD device or even on

the memory bus as a Non Volatile Dual Inline Memory Module (NVDIMM). In this placement, each compute node has access to the full bandwidth of the NVRAM media without any interference from the I/O happening in other compute nodes. However, it may impose an additional data management burden on the user because the files will likely be local to each compute node, i.e. in a private namespace. It is possible that some node-local BB implementations will provide support for multiple compute nodes reading/writing the same file, i.e. shared file I/O. However, it is unlikely to be POSIX-compliant and may be restricted to the case where compute nodes can only read/write to unique subsets of the same file. The second placement is to have NVRAM media in separate I/O nodes between the compute nodes and the storage servers. The advantage of this placement is that it becomes easier to support a shared namespace in which multiple compute nodes can access and consistently update the same files. This design can achieve high performance because the NVRAM is on the HSN and can allow I/O intensive applications to use more than an equal share of the total I/O performance. This is attractive from a system utilization perspective, but has the disadvantage that other applications may have lower peak I/O performance and be subject to I/O interference. Finally, the NVRAM can be placed in the storage servers as a high-speed cache. This has the advantage that the user does not need to make any application or job script modifications to use the NVRAM. However, application I/O performance will be limited by the extra latency of this design and the performance of the SAN. It is also a shared resource and so application data may be evicted from the NVRAM cache because of I/O from other applications. A detailed discussion of the architectural placements is given in [8].

BB Software. The capabilities provided by the BB software differ widely between BB implementations. At a minimum the BB software must provide a mechanism to make the BB storage accessible to user compute jobs. The current generation of BBs do this by creating a POSIX file system over the BB storage. This allows user applications to access storage through the standard POSIX I/O API. The BB software must also provide a mechanism to move data between

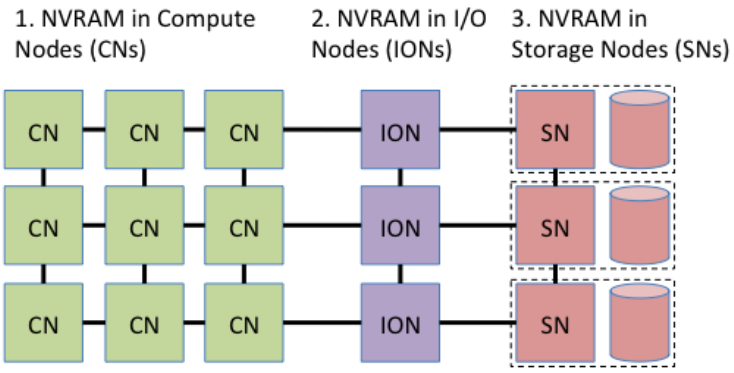


Figure 1: Possible Burst Buffer placements in a supercomputer

the BB tier and the PFS tier. This could be done through an API and/or could be transparent to the user by treating the BB as a cache for the PFS. An API is attractive because data movement costs between tiers can be hidden by moving files at optimal times, however, it imposes an additional burden on the user. A cache is attractive because no code modifications are needed, however, the value of a cache depends on the reuse of file data and read/write mix. At the current time, there is no standard API for BBs. This further increases the burden on users choosing to use an API because different BB implementations will require a different set of API calls. There are ongoing efforts and discussions to develop appropriate standard APIs.

The BB software may be integrated with the site workload manager to enable creation of the file system on the BB at job start up and deletion at job completion. In a shared BB architecture, the underlying BB resource may even be exposed to the site workload manager as a schedulable resource. This allows user compute jobs to request a custom amount of the BB resource. In this scenario, it is desirable for users to request the minimum required BB allocation to minimize queue wait time. The high relative cost of storage is encouraging computing centers to charge for resources other than CPU hours, which further motivates optimizing the BB allocation in jobs.

2.2. The NERSC Burst Buffer Architecture

The NERSC Cori system features a BB based on Cray DataWarp [9]. This is a shared BB design described by placement #2 in Figure 1. There are two flavors of I/O node in the Cori architecture: Lustre network (LNET) nodes to route Lustre I/O traffic and Cray DataWarp nodes (BB nodes) to implement the BB. This design allows users to completely bypass the BB nodes and directly access the PFS through the LNET nodes. The Cray DataWarp nodes contain two Intel P3608 SSDs that deliver 6.4 TiB of usable capacity and 5.7 GiB/s of bandwidth as well as 64 GB of DDR3 DRAM which can be used as page cache. Cori has a total of 288 BB nodes, over 1.8 PiB of usable capacity, and over 1.7 TiB/s of peak performance.

Cray's DataWarp middleware provides user jobs with a dynamically provisioned private PFS over the storage in the BB nodes. Users request a certain capacity of BB in 200 GiB increments (which we call *fragments*) when submitting jobs. The capacity request is then satisfied by placing each 200 GiB fragment on a different BB node so that BB performance scales with the requested capacity. DataWarp designates one of the BB nodes in the allocation as a metadata server. The metadata server manages the file system namespace and stores information about how files are striped over BB nodes. The dynamically provisioned PFS is mounted on the job nodes when the job is launched, and it is typically torn down upon job completion. However, users may also request a *persistent mode* allocation, which allows a BB allocation to persist across multiple jobs.

DataWarp also offers *private mode* reservations where each compute node gets its own metadata server within the BB allocation and, by extension, its own private namespace. This enables higher aggregate metadata performance since each compute node's metadata is serviced by a unique BB node.

DataWarp storage reservations can be configured in scratch mode (`type=scratch`) or cache mode (`type=cache`) [10]. In scratch mode the user is responsible for explicitly moving files between the PFS and DataWarp storage. This can be achieved using job script directives, namely `stage-in` (from PFS to

BB) and `stage-out` (from BB to PFS), or through a DataWarp library named `libdatawarp.a`. The cache mode is designed to abstract away the BB and implicitly move files between BB and PFS when they are accessed. The cache mode capability was not available at the time of our study.

3. Methodology

In this section, we detail our performance analysis methodology and describe the workloads, resources, performance tools and workload configuration used in our experiments.

The objective of our work is to optimize the use of a BB I/O accelerator in a scientific workflow. For this work, we consider a workflow to be a set of applications that have dependencies as represented by a Directed Acyclic Graph. We study the I/O workload, i.e. the data read/write operations and metadata open/close/stat operations, occurring at each stage of the workflow. We describe observed I/O workloads as *bandwidth-bound*, *IOPS-bound* and *metadata-bound* depending on whether I/O time is dominated by large read/write operations, small read/write operations or file open/close/stat operations. We compare the observed I/O workloads against I/O microbenchmarks because we already have a detailed understanding of how to run I/O microbenchmarks optimally on a supercomputer with a BB. We also monitor the reuse of file data and mixture of read/write operations to assess whether different BB software and middleware capabilities can help with the data movement requirements of the workflows.

3.1. Workloads

Our evaluation uses a combination of two scientific workflows and I/O benchmarks to understand the performance characteristics of the workflows and peak I/O performance of the BB.

3.1.1. I/O Benchmarks

Interleaved Or Random (IOR). The IOR [11] I/O benchmark is widely used in HPC to measure peak storage system performance. IOR is an MPI

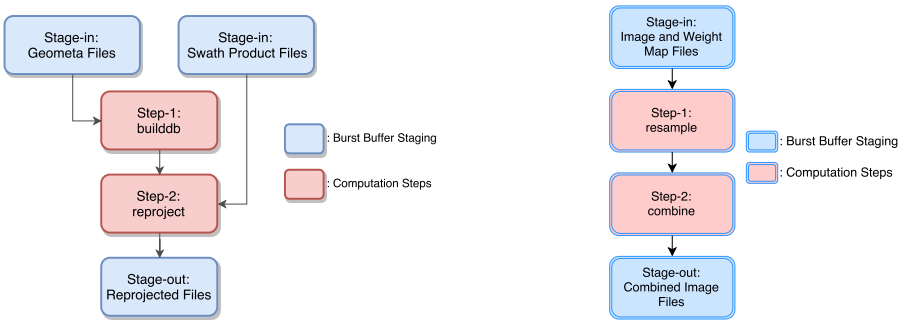
application which measures read / write performance when using POSIX, MPI-IO and other interfaces. Many aspects of the I/O can be varied including the access pattern, transaction size and whether to use a shared file or file per MPI rank. In this paper we select two IOR configurations to demonstrate the peak data transfer bandwidth and IOPS of the BB. The IOR configurations use the POSIX I/O interface, one file per MPI rank and a sequential access pattern because structured I/O patterns tend to appear more often in HPC. We define a structured access pattern as a regular access pattern, e.g. sequential or strided. These patterns occur when reading contiguous (i.e. sequential) and non-contiguous (i.e. strided) slices of a multi-dimensional array.

MDTest. The MDTest [12] I/O benchmark is an MPI application which measures the performance of storage system metadata operations. The benchmark creates empty files, runs `stat` on the files and then removes the files. In this paper we use MDTest to obtain the peak metadata rate for `create`, `open`, `stat`, and `unlink` metadata operations.

3.1.2. Workflows

The two workflows studied in the paper are selected because they stress the I/O subsystem in different ways: CAMP is limited by metadata performance and SWarp is limited by data transfer performance. In addition, CAMP has different data usage and movement pattern as compared to SWarp. This allows us to evaluate different performance characteristics of the workflows on the BB. When discussing the workflows, we use the term "workflow pipeline" to refer to a single unit of the larger workflow.

CAMP. The CAMP (Community Access MODIS Pipeline) workflow processes Earth's land and atmospheric data obtained from MODIS satellite data [13, 14, 15]. It transforms the MODIS data from a swath space and time coordinate system (latitude and longitude) into a sinusoidal tiling system (tiles using sinusoidal projection). The MODIS data for CAMP consists of small geometa files in plain text format and swath products as Hierarchical Data Format (HDF) files. Each geometa file is only a few KBs and is used by all the swath prod-



(a) CAMP (b) SWarp

Figure 2: Science workflows. a) CAMP has data staging operations to move the data from the parallel file system to the BB and vice-versa, and has two compute stages, builddb and reproject, to transform the swath products to a sinusoidal tiling system. b) SWarp has data staging, resample and combine stages to produce high quality reference images. Both workflows have different data usage and access patterns.

ucts from a particular satellite. Each swath product has several files per day, each of which is approximately 1.1 MB in size and contains the product data in swath space and time coordinate system. The conversion of a product from swath coordinate to a sinusoidal tile requires several of these swath product files, along with the mapping information derived from the geometa files. Hence, the CAMP workflow consists of two processing steps – a) **builddb**, that assembles and maps swaths to their corresponding sinusoidal tiles and b) **reproject**, that converts the MODIS products from a swath coordinate system to a sinusoidal tiling system. Figure 2a shows the high-level representation of the CAMP workflow that includes the data staging operations to and from the BB.

CAMP is written in Python and uses an SQLite database to store the swath to sinusoidal tile mapping. For our analysis, we use each task in the CAMP workflow to transform one MODIS product's swath coordinates for one day into one specific tile. Hence, the builddb step generates one SQLite database per task and the reproject step uses the mapping stored in the database to transform the swath coordinates into a sinusoidal tile. The final outputs of the reproject step are HDF files for each product and each tile for a selected day. Each output

file is a few MBs in size. For our experiments, we run identical tasks (for the same product, day and tile) across multiple compute nodes with one task per node. The steps of the workflow are run one at a time to understand the I/O characteristics of the workflow for each step. We use Conda, which uses the Anaconda Python distribution, to install CAMP on DataWarp.

SWarp. The SWarp workflow combines overlapping raw images of the night sky into high quality reference images. It is used in the Dark Energy Camera Legacy Survey (DECaLS) to produce high quality images of 14,000 $\deg^2$ of northern hemisphere sky. In this survey, each SWarp workflow pipeline produces an image for a 0.25 $\deg^2$ "brick" of sky. The average input to each workflow pipeline is 16×32 MiB input images and 16×16 MiB input weight maps. The entire workflow consists of nearly one hundred thousand SWarp workflow pipelines, which are embarrassingly parallel. However, even though the computation is trivially parallelizable, the same input image often overlaps with multiple bricks.

The SWarp workflow pipeline consists of a data resampling stage and a data combination stage as shown in Figure 2b. The data resampling stage interpolates the raw images so that the images can be trivially stacked. A resampled image is created for each raw image. The data combination stage reads back the resampled images and then performs a reduction over the pixels to produce a single stacked image. The raw, resampled and stacked images are all in Flexible Image Transport System (FITS) file format. The DAG when using a BB is similar to CAMP: input images and weight map files are staged-in prior to the data resampling stage and the combined image is staged-out after the data combination stage. A single executable named `swarp` implements both stages. The stages can be launched independently or run as part of an automated workflow. In this paper, we run the stages one at a time so that we can collect I/O characteristics and scaling data per workflow stage. SWarp is written in C and multithreaded with POSIX threads. The multithreading strategy involves different threads operating on different regions of the same image. All threads are used in the resampling stage and then all threads are used in the co-addition stage.

3.2. Machine Setup

The experiments were performed on the Cori supercomputer at NERSC. Cori is a Cray XC40 with two partitions: Cori Phase 1 consists of Intel Xeon® E5-2698 v3 ("Haswell") processors and Cori Phase 2 consists of Intel Xeon Phi™ 72-50 ("Knights Landing") processors. The BB described in Section 2.2 is connected via the Cray Aries network and is accessible to both partitions. We only used the Cori Phase 1 partition because it was the only partition available at the time of result collection. Cori Phase 1 consists of 2388 nodes with 128 GiB DRAM memory per node. Each node consists of two 16-core Haswell processors nominally clocked at 2.3 GHz (turbo up to 3.6 GHz).

3.3. Performance Tools

The performance tools used to collect workflow characteristics and timing information are Strace-4.12 [16] and IPM-2.0.3 (revision 16c494310b) [17].

Strace. Strace intercepts system calls from a compiled application or script. It is able to provide detailed information about each system call including arguments, return values and duration of system calls. We use it to get an accurate picture of the I/O performed at each stage of a workflow. Intercepting at the application level can be misleading because streaming, character and formatted I/O are all buffered by the C library (`libc`). For example, an application reading data with many sequential `fread` operations of size 1 KiB may actually read the data with a smaller number of `read` system calls of size 8 MiB. Since our work is concerned with application interaction with storage, we are more interested in the kernel issued I/O calls. The information we collect with Strace are I/O system call counts, read and write transfer sizes, amount of I/O and estimates of file sizes.

IPM. The Integrated Performance Monitoring (IPM) profiling tool [17] provides performance information about application computation, memory usage, communication and I/O. The I/O monitoring infrastructure in IPM works by providing a collection of functions which wrap `libc` I/O calls, e.g. `fread` and `read`, at link or run time. The wrapper functions record the time to execute the

underlying `libc` I/O function as well as function specific information such as bytes requested, bytes transferred and file offsets. Darshan [18] also wraps `libc` I/O calls and is installed at many supercomputing centers. It is not appropriate for our work because it is limited to MPI applications and does not measure child processes' I/O. We need support for serial applications to evaluate the two workflows.

It is important to note that wrapping `libc` I/O functions does not catch all application I/O because system I/O calls can be made directly without first going through `libc`. For example the dynamic loader `ld-2.19.so` loads many shared libraries at program start-up in this way. Despite IPM missing some I/O, a comparison of IPM and Strace measurements shows that we are capturing all of the significant I/O from these workflows.

3.4. Workload Configuration

The workflow pipelines are run on Cori with the production configuration thread count. We stage the input data into the BB by placing `stage-in` directives in the job script before running the workflow pipelines. All I/O is directed to the DataWarp mount points and no data is staged out from the BB to the PFS. The DataWarp reservation is configured to use a single fragment of capacity. We use a job reservation for SWarp and a persistent reservation for CAMP. We choose to use a persistent reservation for CAMP so that multiple CAMP jobs can reuse the Python software environment in the BB. The alternative would have been to install the Python environment for every single CAMP job but this is computationally expensive.

The workflow pipelines are run on a single compute node with Strace monitoring. This allows us to collect I/O characteristics, e.g. amount of I/O, I/O transaction size and I/O call count, describing the kernel issued I/O requests. We are able to derive the average reuse of data and percentage of I/O to user data files from the Strace data. We then repeat the same run with IPM monitoring to measure the time in different I/O operations. We do not use the two types of monitoring at the same time because Strace has high overhead and

would significantly increase the time spent in I/O. This approach allows us to derive bandwidth, IOPS and metadata rates delivered by the storage system without being affected by the buffering happening within `libc`.

The scaling performance of the workflows is then investigated to find out at what point workflow runtime is affected by saturation of I/O. The workflow pipeline is replicated on 1 to 64 compute nodes and I/O is directed to a fixed storage reservation of 1 DataWarp fragment. We place a single process per node because we strive for constant compute time per node: adding more processes per node would share network injection bandwidth and potentially memory bandwidth and last level cache resources on each socket. The batch script which runs the workflow pipeline(s) consists of an `srun` launch line per compute node. Each `srun` executes a thin wrapper script on each compute node which delays execution of the actual workflow pipeline until a control file is detected in the PFS. This is necessary because the processes may otherwise begin several seconds apart. The wrapper script polls for the control file every 0.01 seconds which is eventually created by the batch script 10 seconds after all `srun` launches. The performance is measured with IPM. Strace is not needed because the I/O characteristics of replicated workflow pipelines are the same.

4. Results

In this section we show performance results relevant to the execution of the workflows on a BB. Section 4.1 outlines the baseline performance of Cori's BB to provide context for workflow performance results. Section 4.2 shows the performance of the workflows using Cori's BB in both single pipeline and multiple pipeline configurations. Section 4.3 outlines a strategy to optimize metadata-bound workloads on DataWarp BBs. Section 4.4 considers the workflows' data movement and the capabilities needed from BB software. Finally, Section 4.5 summarizes the key results.

4.1. Cori Burst Buffer Performance

Figure 3 shows how the bandwidth of the storage allocation changes with I/O concurrency. The results were obtained by running the IOR benchmark with a large transaction size of 8 MiB against a single fragment of DataWarp. The benchmark is run in a strong scaling configuration with 1 MPI rank per node and a fixed aggregate file size. The aggregate file size is chosen to be 128 GiB in order to exceed the BB node memory capacity and avoid server-side caching. There is no possibility of data being cached by the compute node because the client-side caching feature of DataWarp (`client_cache=yes` [10]) was not available at the time of the study. The results show that a single I/O thread achieves 2 GiB/s read bandwidth and 1.5 GiB/s write bandwidth. The highest performance of 6.5 GiB/s read bandwidth and 5.3 GiB/s write bandwidth is only achieved when using at least 8 I/O threads. There is no benefit to using more than 8 I/O threads in this configuration because the SSDs on the BB node are saturated.

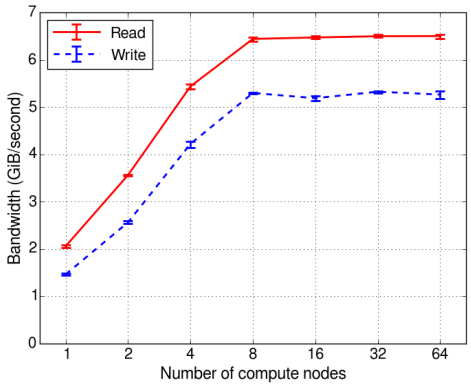


Figure 3: IOR bandwidth at 8 MiB transaction size

Figure 4 shows how the IOPS performance of the storage allocation changes with I/O concurrency. The IOPS metric is chosen rather than bandwidth to emphasize the peak throughput of small I/O transfers. The same IOR configuration is used as before except that the transaction size is 4 KiB (2,048 times

smaller than the first configuration). We use a smaller aggregate file size of 1 GiB because an IOPS-bound configuration takes much longer to run. We additionally run IOR with the `fsyncPerWrite` option which performs a `fsync` operation after each write transaction. This flushes data from DRAM to the SSDs and ensures we measure storage performance. It is helpful because the aggregate file size is smaller than the capacity of BB node DRAM and can potentially benefit from server-side caching. The results in Figure 4 show that 16 I/O streams with no synchronization achieves approximately 94K read IOPS and 85K write IOPS. For comparison with the earlier plot, 94K IOPS at 4 KiB transaction size is 0.36 GiB/s. The write IOPS are approximately halved when using synchronization because there is overhead associated with every I/O request and the data is guaranteed to be flushed to the SSDs.

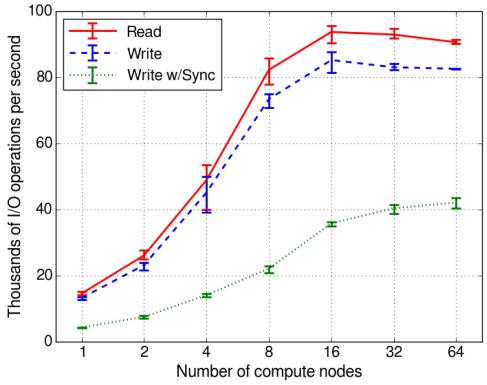


Figure 4: IOR IOPS at 4 KiB transaction size

Figure 5 shows how the metadata performance of the storage allocation changes with concurrency. The results are obtained by running the MDTest benchmark with 1 MPI rank per node against a single DataWarp fragment. We run the benchmark in a strong scaling configuration in which the total count of files is kept constant. The file count is chosen to be 65,536 files so that the benchmark runs long enough to collect reliable data. The figure shows throughput for the Stat and Remove operations and pairs of Open/Close and Create/Close

operations. The operation throughput versus concurrency is similar for Stat, Remove and Open/Close, and peaks at approximately 25K operations per second at 32 MPI ranks. The Create/Close operation throughput is less, indicating that file creates are a more expensive metadata operation.

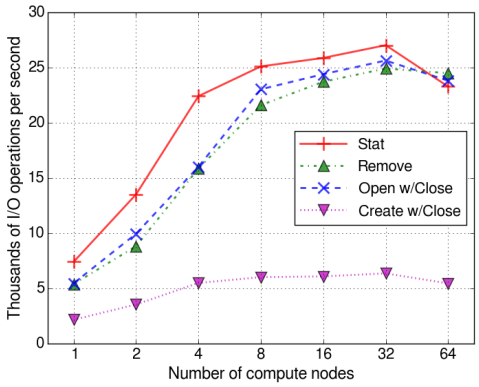


Figure 5: MDTest operation throughput

The BB performance plots have only considered I/O originating from the Intel Xeon® CPUs in Cori Phase 1. We consider the work of Liu et al. [19] to explain how I/O performance will be different when I/O originates from the Intel Xeon Phi™ CPUs in Cori Phase 2. Firstly, single-stream I/O bandwidth will be up to 3.5x lower than Xeon®. This is because of lower clock speed and increased overhead of managing the kernel page cache. Secondly, up to 48 I/O streams will be needed to reach the peak I/O performance of a single fragment of the BB. The implications of this are considered in Section 5.1.

4.2. Workflow Performance

4.2.1. Performance of a Single Workflow Pipeline

Table 1 shows the resource requirements of the individual workflow stages. The names of the workflow stages are abbreviated to save space: SWarp-resample, SWarp-coadd, CAMP-builddb, CAMP-reproject map to SWarp rsmpl, Swarp coadd, CAMP db, CAMP reprj, respectively. The memory footprint is

approximately 100 MiB for all workflow stages except for SWarp-coadd which has a memory footprint of approximately 1 GiB. Although our experiments only ever use 1 process per compute node, this indicates it is possible to run many concurrent workflow pipelines on a single compute node of Cori Phase 1 (which has 128 GiB of memory). The table also shows the total size of all files accessed during each workflow stage. We add together the totals to give an upper-bound estimate of the end-to-end workflow pipeline storage requirements. The results indicate that a single workflow pipeline requires significantly less storage than is provided by a 200 GiB fragment of DataWarp. For example, CAMP only has a storage requirement of 151.6 MiB. This means that up to 1350 CAMP pipelines could run concurrently against the same unit of DataWarp storage.

	SWarp rsmpl	**SWarp coadd**	**CAMP db**	**CAMP reprj**
Compute threads	16	16	1	1
I/O threads	1	1	1	1
Peak memory footprint (MiB)	108.8	1064.7	96.1	93.0
Total file size (MiB)	1686.5	1016.8	74.1	77.5
Total file size per pipeline (MiB)		2703.3		151.6
Max pipelines per DataWarp fragment		75		1350

Table 1: Resource requirements of a single workflow pipeline

Table 2 shows the performance of the individual workflow stages. The table consists of three result groups: time measurements, I/O characteristics, and derived performance metrics. The workflow stages spend between 10% and 30% of time in I/O. The time consists of time spent in data operations and metadata operations. The results show that SWarp workflow stages spend longer in data operations and that CAMP workflow stages spend longer in metadata operations. The I/O is significantly different between the two workflow pipelines. SWarp performs several GiBs of I/O with relatively large average transfers of more than 3 MiB. Large transfers are needed to reach the bandwidth performance peak of the underlying storage hardware. CAMP performs less than 100

MiB of I/O with relatively small transfers of 20 KiB or less. The average transfer size in CAMP-builddb is only 3 KiB because the SQLite library performs many small updates to the database file and journal. Small transfers stress the storage hardware differently and the peak IOPS value becomes more relevant than peak bandwidth. The total count of metadata operations are almost two orders of magnitude higher in CAMP than SWarp. The large number of metadata operations arise from Python module imports and frequent file system interactions by SQLite.

	SWarp rsmpl	**SWarp coadd**	**CAMP db**	**CAMP reprj**
Wall time (s)	10.7	4.7	15.3	9.2
I/O time (%)	20.3	26.0	13.5	16.6
I/O time in data operations (s)	1.87	0.97	0.78	0.57
I/O time in metadata operations (s)	0.30	0.26	1.28	0.95
I/O (MiB)	2711.5	1422.8	25.3	34.7
Read/write operations	784	430	7413	1735
Mean read/write transaction size (KiB)	3542	3388	3	20
Metadata operations	413	318	11963	7820
Bandwidth (MiB/s)	1452	1465	32	61
% of peak bandwidth	23.6	23.8	0.5	1.0
Read/write rate (operations/s)	420	443	9467	3060
% of peak IOPS	0.5	0.5	10.5	3.4
Metadata rate (operations/s)	1381	1237	9324	8240
% of peak metadata rate	5.5	4.9	37.3	33.0
Pipelines to saturate data performance	5	5	10	30
Pipelines to saturate metadata performance	19	21	3	4

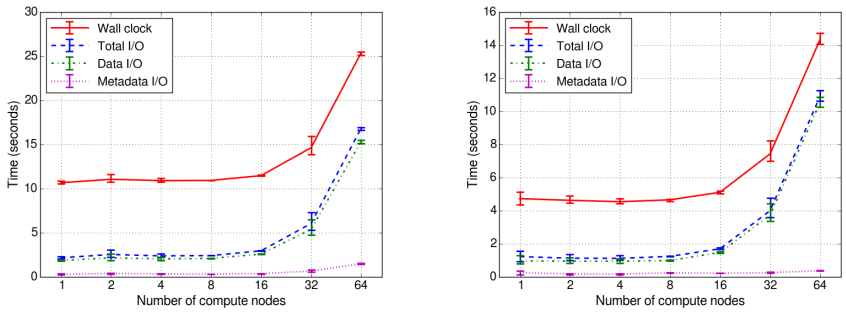
Table 2: Performance of a single workflow pipeline using DataWarp. The table also estimates the percentage of peak I/O performance and the number of workflow pipelines to saturate storage

The final group of values in the table show the attained performance relative to the storage performance peak. We estimate the peak values from the I/O benchmarks to be 6 GiB/s bandwidth, 90K IOPS, and 25K metadata operations per second. The results show that no workflow stage gets close to the peak per-

formance of the BB allocation in any performance dimension. SWarp transfers data at over 1 GiB/s/pipeline which is over 20% of the storage configuration peak performance. CAMP transfers data at less than 1% of peak bandwidth, however, the small transfer size makes peak IOPS the more relevant performance upper-bound. CAMP-builddb is found to transfer data at more than 10% of the peak IOPS for this storage configuration. The CAMP workflow stages are also much more metadata intensive and are driving the storage configuration at more than 30% of peak. Finally, we use the percentage of peak to estimate the number of concurrent workflow pipelines to saturate the data and metadata performance of this storage configuration. The data peak is obtained by taking the maximum of the bandwidth and IOPS percentage of peak. The results show that both SWarp workflow stages are expected to saturate the data component of I/O when using only 5 concurrent workflow pipelines. CAMP is expected to saturate the metadata component of I/O when using only 3 (CAMP-builddb) or 4 (CAMP-reproject) workflow pipelines. The number of concurrent workflow pipelines to saturate performance are significantly less than the number needed to saturate storage capacity, indicating that reducing the system fragment size may be a beneficial optimization for these workflows.

4.2.2. Performance of Concurrent Workflow Pipelines

The time in I/O for a single workflow pipeline represents the best achievable time and can only get worse as more workflow pipelines contend for the same storage resource. We show how the workflow stages scale when increasing the number of workflow pipelines. Ideal scaling will only happen if the single DataWarp fragment can sustain the aggregate I/O requests. We show a plot of time per workflow pipeline versus concurrency for each workflow stage. The time quantities of interest are wall clock time, I/O time, I/O time in data operations and I/O time in metadata operations. The experiments are repeated three times at each node count and the plots show the mean time per workflow pipeline stage. The error bars show the minimum and maximum time in the three trials and are helpful for capturing the performance variability when using the Cori



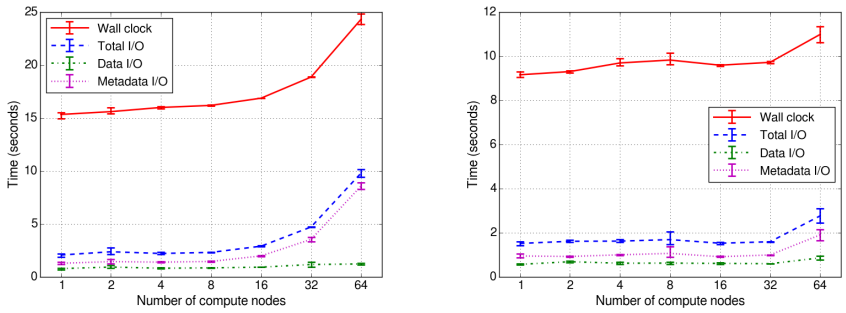
(a) SWarp-resample (b) SWarp-coadd

Figure 6: Scaling of SWarp workflow with number of workflow pipelines

supercomputer.

Figure 6 shows the scaling of the SWarp workflow. The scaling of SWarp-resample is similar to SWarp-coadd because both stages use the same I/O kernel to transfer image data. The results show that wall clock time remains relatively constant until approximately 16 concurrent workflow tasks, which is a greater degree of scaling than the earlier bandwidth-bound IOR benchmark configuration (see Figure 3). It happens because the individual SWarp-resample and SWarp-coadd workflow tasks are only synchronized at launch time and so independent I/O requests from concurrent tasks are not necessarily happening at the same time. The scaling loss is more significant for SWarp-coadd at higher concurrencies because a single SWarp-coadd task spends a greater fraction of time in I/O. The I/O time is dominated by data rather than metadata operations in both workflow stages. Metadata time increases slightly at 64 pipelines for SWarp-resample because SWarp-resample operates on more data files than SWarp-coadd.

Figure 7 shows the scaling of the CAMP workflow. Both CAMP workflow stages spend most of the I/O time servicing metadata operations. One source of these metadata operations is from starting Python applications. This metadata load happens because Python searches for files providing a package in every directory in the Python path. It is known to be a scalability issue in Python



(a) CAMP-builddb (b) CAMP-reproject

Figure 7: Scaling of CAMP workflow with number of workflow pipelines

HPC applications [20]. Interestingly, the metadata time increases by 7-8 seconds in CAMP-builddb when increasing the concurrency from 1 to 64 workflow pipelines, but only increases by 1 second in CAMP-reproject. This cannot be explained by differing Python metadata cost because the Python path is the same and the Python dependencies are nearly identical in both workflow stages. The source of the additional metadata operations in CAMP-builddb are from saving data to an SQLite database. This saturates the metadata server when using 64 workflow pipelines, as indicated by the metadata time increasing by more than a factor of two when increasing from 32 to 64 workflow pipelines. The time in data operations is relatively constant for both workflow stages at all tested concurrencies.

Figure 8 shows the cost of running the workflow stages at different concurrencies. We calculate relative wall clock time by dividing average wall clock time for multiple workflow pipelines by wall clock time for 1 workflow pipeline. The relative wall clock time remains close to 1.0 up until 16 workflow pipelines. There is a greater degree of performance loss for SWarp than CAMP beyond 16 workflow pipelines because the impact of saturation is more pronounced when applications spend a greater fraction of time in I/O.

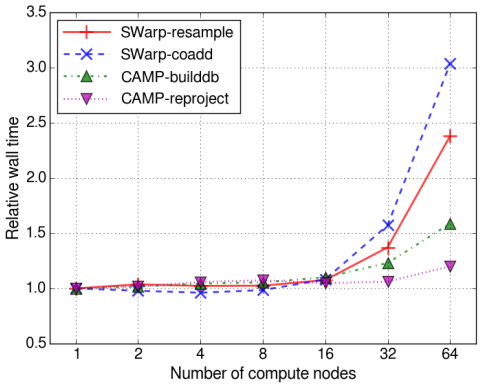


Figure 8: Relative wall clock time per workflow pipeline

4.3. Optimization of CAMP-builddb

The CAMP-builddb workflow stage is interesting because the metadata time increases rapidly with concurrency. This bottleneck cannot be overcome by simply adding DataWarp fragments because this would only address the data component of I/O. The high I/O overhead exists because each SQLite database transaction involves many POSIX I/O operations [21]. One way to reduce I/O overhead is to place multiple database operations within a larger transaction [22], however, this optimization may be out of reach for most end users. Another approach is to relax data consistency for the application as a whole by setting certain SQLite pragmas [23]. This includes `synchronous=OFF` to eliminate synchronizations to the database file. It also includes `journal_mode=MEMORY` to use in-memory journaling which removes many metadata operations, including creating, stating, closing and removing the journal file every transaction.

Figure 9 shows time in I/O for several permutations of CAMP-builddb run on 64 compute nodes. Each box in the plot shows the interquartile range (IQR) of the data and the box whiskers show the most extreme data points within $1.5 \times IQR$ from the lower and upper quartiles. A horizontal line within each box, a square marker, and plus markers indicate the median, mean and outlier data points, respectively. "Default #1" and "Default #2" indicate the origi-

nal configuration of CAMP-builddb. "No sync." indicates a modified version of CAMP-builddb with the pragma `synchronous=OFF`. "No sync. and mem. journal" is the same as "No sync." with the addition of the pragma `journal_mode=MEMORY`. Finally, "Private mode" indicates the original configuration of CAMP-builddb run against a private namespace storage allocation. The allocation is configured with a capacity of 12,800 GiB (200 GiB $\times$ 64) so that the number of metadata servers is equal to the number of compute nodes.

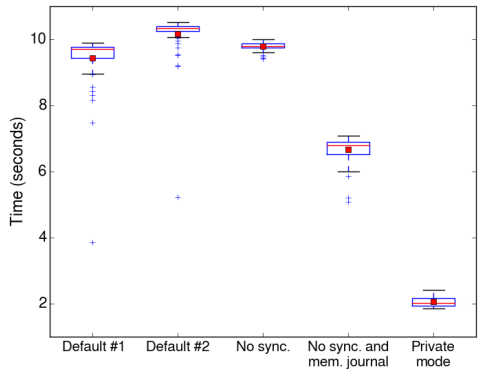


Figure 9: Box plot of time in I/O for various configurations of CAMP-builddb on 64 compute nodes

The results show that "Default #1" and "Default #2" spend approximately 10 seconds in I/O. The time does not change noticeably for the "No sync." version because only a small amount of time is spent in the `fdatasync` synchronization operation. This reinforces our earlier observation that CAMP-builddb is metadata-bound. We mention this configuration because it would improve performance significantly on a storage system with spinning disk. The "No sync and mem. journal" configuration reduces I/O time by approximately 1.5x indicating that it is possible to reduce I/O time by sacrificing data safety in SQLite. Finally, "Private mode" reduces I/O time significantly by approximately a factor of 5. This indicates that users have a way to run unmodified metadata-bound applications on DataWarp at scale without suffering performance penalties.

4.4. Data Movement Characteristics

Table 3 shows the data movement characteristics of the workflow pipelines. The first row in the table shows the percentage of I/O to files which are relevant to the user's scientific investigation, e.g. images, databases and HDF5 files. Comparison of total I/O to scientifically meaningful I/O (User I/O) shows that 100% of SWarp I/O is associated with scientific data files (at least to 1 decimal place of precision). This is in contrast to CAMP-builddb and CAMP-reproject which have values of only 23.3% and 41.3%. The source of the missing I/O is the reading of Python packages. This is only a modest amount of I/O but is much larger than the I/O to the small scientific data files. The next two rows in the table give information about the average reuse of data for each workflow stage. They are calculated by dividing the amount of I/O by the file size. The average reuse of data is extremely low for both SWarp workflow stages because the input files are read a maximum of two times and output files are only written once. A trivial optimization of avoiding the repeated read (possible because the input files are completely read into memory) would further reduce the average reuse of data. The average reuse of data is less than 1.0 for CAMP because only 2 KiB of data is read from Intel Math Kernel Library (MKL) files of size 10 to 25 MiB. If MKL is excluded then the average reuse is close to unity. CAMP-builddb has a high value of user data reuse because of the repeated updates to the SQLite database file and journal file. The final two rows of the table show the read to write ratio of the workflow stages. SWarp-coadd is dominated by reads because multiple resampled images are combined into a single high quality image. The reading of Python packages by CAMP contributes to higher read to write ratios of 2.9 and 12.7 for CAMP-builddb and CAMP-reproject, respectively. CAMP-reproject is more dominated by reads than CAMP-builddb because the user HDF5 files are read multiple times.

4.5. Summary

- The I/O characteristics of the SWarp and CAMP workflows are very different. SWarp is more likely to be limited by peak storage bandwidth

	SWarp rsmpl	**SWarp coadd**	**CAMP db**	**CAMP reprj**
User I/O (% total)	100.0	100.0	23.3	41.3
Data reuse	1.6	1.4	0.3	0.4
User data reuse	1.6	1.4	13.9	2.7
Read:write I/O	2.0	13.4	2.9	12.7
Read:write user I/O	2.0	13.4	0.6	9.9

Table 3: Data movement requirements of a single workflow pipeline

because of large transactions of mean size 3-4 MiB, where as CAMP is more likely to be limited by peak IOPS because of small transactions of mean size 3-20 KiB. CAMP also performs many more metadata operations than SWarp [Section 4.2.1].

- The I/O load of multiple workflow pipelines saturate different aspects of the storage system. SWarp saturates data transfer performance at higher concurrencies. CAMP does not saturate data transfer performance even at high concurrency indicating that the BB storage is able to sustain the aggregate IOPS. CAMP-builddb (and to a lesser extent CAMP-reproject) saturates metadata performance at higher concurrencies [Section 4.2.2].

- The results indicate that configuring and allocating the BB as per the needs of the individual stages of the workflow may yield better performance than having a single storage allocation and configuration for the entire workflow (e.g. private mode configuration for CAMP-builddb) [Section 4.3].

- The workflows are generally dominated by reads of both application data files and Python modules. This indicates that the checkpoint-restart use case is not a good proxy for the I/O in data analytics workflows. It also highlights the value of explicitly staging both types of data into the BB by workflow systems before running the applications [Section 4.4].

5. Discussion

In this section, a) we discuss the key characteristics of the workflows analyzed and use the information to highlight the effective use of BBs, b) we apply this knowledge and explain how to achieve optimum performance with the DataWarp implementation of a BB and, c) we discuss limitations with the DataWarp APIs in context of the storage needs of the workflows.

5.1. Efficient Use of Burst Buffers

The key findings from our experimental analyses are:

- **A single workflow pipeline does not provide the I/O parallelism needed to make efficient use of BBs.** The data analytics workflows studied in this paper consist of single-process applications which perform I/O with a single thread of execution. This is poorly matched with the need to have multiple I/O streams to obtain the peak performance from BB flash storage. Unfortunately, single I/O stream workflow pipelines are a common feature of high throughput data analytics workflows. We show that better utilization of BB resources is possible by executing multiple concurrent workflow pipelines against the same unit of BB storage. Our results indicate that a single unit of DataWarp storage on Cori can sustain the I/O requests from approximately 16 concurrent workflow pipelines before there is any slow down. The performance of single I/O stream workflow pipelines are worse when using Xeon Phi™ rather than Xeon® CPUs. Therefore, it becomes even more important to provide sufficient I/O parallelism when using Xeon Phi™ CPUs.

- **The I/O performance limiter at scale can be determined by analyzing the I/O from a single workflow pipeline.** The peak bandwidth, IOPS and metadata performance of any BB system can be obtained by using our I/O microbenchmark configurations. The I/O workload of the workflows can then be compared against the appropriate microbenchmark configuration to estimate the number of workflow pipelines needed

to saturate storage. We predicted that SWarp would saturate data performance at five workflow pipelines and CAMP would saturate metadata performance at three or four workflow pipelines. The analysis correctly identified whether data or metadata bottlenecks are more significant at scale. However, the predicted values were less than the empirical values of around 16 for two reasons. First, the workflows spend less than 30% of time in I/O and so I/O saturation has less effect on overall run time at relatively low concurrencies. Second, the workflow pipelines are loosely-coupled and so the I/O requests from independent workflow pipelines do not necessarily happen at the same time. This is in contrast to the I/O microbenchmarks which are tightly-coupled MPI applications.

- **A scaled out workflow pipeline is often limited by metadata performance.** Our analysis has found significant metadata costs originating from database transactions, Python initialization and opening many small files. The aggregated metadata operations from multiple workflow pipelines can easily saturate a single metadata server, as shown in the CAMP-builddb workflow stage. We have demonstrated that CAMP-builddb metadata overhead can be reduced significantly by relaxing data consistency and restricting file visibility via private DataWarp namespaces per compute node.

5.2. Efficient Use of DataWarp

DataWarp storage reservations consist of multiple storage fragments of a fixed size. The fragment size is configured to be 200 GiB on Cori. The optimal amount of storage to reserve depends on the capacity and performance needs of the workflow. In this paper, we have found that both SWarp and CAMP are limited by DataWarp performance rather than capacity. SWarp and CAMP have an aggregate capacity requirement of up to 2.6 GiB and 150 MiB per workflow pipeline, respectively (Table 1). However, the performance saturates before fully utilizing the 200 GiB of capacity at approximately 16 workflow pipelines per DataWarp fragment (Figure 8). This means that excess capacity must be

reserved for both workflows to sustain performance on Cori, and reducing this fragment size would allow similar workflows to make better overall utilization of the BB.

The SWarp workflow is characterized by sequential reads and writes consisting of transactions with a mean payload of a few Megabytes. Workflows limited by data transfer performance will scale out by simply reserving more DataWarp fragments (assuming the fragments are on different DataWarp server nodes). A scientist running SWarp should reserve one DataWarp fragment for every 16 workflow pipelines. CAMP has a larger number of data and metadata operations, and is ultimately limited by metadata performance. Metadata performance can be improved by using a private namespace configuration because this provides a metadata server for each DataWarp node in the storage reservation.

The I/O characterization and time measurements of a single workflow pipeline in Section 4.2.1 are a predictor of aggregate I/O load. This is sufficient to understand whether the scaled out workflow pipeline is going to be bandwidth-bound, IOPS-bound or metadata-bound, and helps us choose the optimum DataWarp storage reservation.

5.3. Improving Burst Buffer APIs

The workflows in this paper are bound by performance rather than capacity of DataWarp fragments. In our experiments, we showed the capacity that needs to be reserved to meet the performance needs of the workflow pipelines. This indirect way of reaching a performance target indicates that the DataWarp API is using the wrong basic building block. The right building block should be performance targets for bandwidth, IOPS and metadata performance. For example, SWarp needs a bandwidth building block and CAMP needs a metadata building block. (Although not covered in this work, a near real-time workflow would probably also request a performance guarantee.)

The private namespace feature of DataWarp enabled us to overcome a metadata scaling bottleneck in CAMP-builddb. However, it is not, as it stands today, the optimal solution because it made it tricky to benefit from improved meta-

data performance for the Python initialization. We did not find a way to create a relocatable Conda Python environment. Thus, we installed the software environment in 64 separate namespaces before running the workflow. Ideally, the DataWarp API should allow us to install the software environment into a persistent reservation once and be able to choose a flexible number of metadata servers.

5.4. Data Management Strategies

Our results show that different workflows have different I/O requirements, and hence, different usage patterns of the BB. The data management solutions need to consider various trade-offs between I/O performance, data and metadata operation overheads and usage patterns of the workflows to provide algorithms and strategies for efficiently managing the data on and across the BB. The strategies also need to consider the structure of the workflows in order to optimize the data reuse and distribution strategies. Some of the key findings of our analysis are:

- **It is valuable to explicitly control the data in the BB tier.** The individual workflow stages are found to have a read to write ratio of 2.0 to 13.4 and an average data reuse of 0.3 to 1.6. The dominance of reads and low average data reuse implies that input files should be ready to read when the workflow pipeline starts. Therefore, we do not expect automatic file movement between the BB and the PFS to benefit these two data analytics workflows. This is because the one-time cost of staging the data at access time cannot be hidden by significant data reuse. In addition, the workflow pipelines consists of a number of intermediate files which can be discarded once there are final results. For example, the resampled images in SWarp and the SQLite database in CAMP are not needed by the scientist. Automatic file movement would cause these files to be transferred to the PFS unnecessarily.

- **It is valuable to leave data in the BB tier for longer than a single**

batch job. We have found that input files and software environments are reused across workflow pipelines.

- The input data for data analytics workflows are generally Write Once Read Many times (WORM). In the SWarp workflow a single input image often contributes to multiple regions of the sky. Therefore it is wasteful to re-stage the same input file multiple times for each workflow pipeline.
- The software environment is reused in every single workflow pipeline. In the CAMP workflow the Python environment is responsible for more than half the total I/O. The role of "support I/O" (e.g. Python packages) is rarely mentioned in the context of BBs. It is useful to stage the software environment once to avoid the overhead and wear of repeatedly staging the software environment.

Long-term data residency is not a good fit for today's BBs because they do not provide data redundancy. This imposes a data management burden upon the developer.

6. Related Work

Scientific Workflows. Data-intensive scientific workflows have been shown to process large amounts of data with varied I/O characteristics [15, 24, 25, 26]. Deelman et al. [27] highlights several challenges in data management for data-intensive scientific workflows, including data storage, movement and metadata management. Several strategies have been proposed to optimize data management for scientific workflows in HPC environments that include just-in-time staging and heuristics to minimize data movement [28, 29, 30]. However, BBs add another layer in the storage hierarchy, adding to the data management challenges for scientific workflows. Ghoshal et al. [31] propose data management strategies on tiered storage with BBs, based on the data usage and access patterns in scientific workflows. In this paper, we evaluate two workflows with

different I/O characteristics and find both workflows are read-dominated with low data reuse. They would therefore benefit from early staging of both data and software libraries on architectures with a BB. This capability would ideally be provided by next-generation workflow systems which would support staging of this data asynchronously and would also optimize task scheduling based on data already resident in the BB.

Burst Buffers. Several uses of BBs have been shown in order to mitigate the I/O bottlenecks of data-intensive workloads [6, 32, 33, 34]. Most studies surrounding the design and use of BBs have so far focused on the I/O characteristics of individual applications [35] or small components within workflows [4]. However, research into optimizing scientific workflows with diverse I/O and storage requirements for BBs is still in its infancy, and a limited body of work presently exists [3, 36]. Beyond single applications and workflows, researchers are investigating I/O-aware scheduling on systems with a BB. Herbein et al. [37] demonstrate that system utilization can be improved by using application drain bandwidth between the BB and PFS as a scheduling constraint. Thapaliya et al. [38] et al. investigate interference issues on a system with a shared BB. They find that runtime scheduling is valuable when independent applications concurrently access the same storage resource. Our work provides the guidelines to improve utilization of a BB. If the guidelines are widely followed by many users this can help reduce interference issues in supercomputer architectures with a shared BB.

DataWarp. DataWarp is Cray's implementation of a BB, and few guidelines exist for how to use it optimally for scientific workflows. Bhimji et. al show performance results for a collection of applications selected as part of NERSC's Early User Program [39]. The results focus on application I/O bandwidth on DataWarp and the PFS. The NERSC website provides a list of known issues and overall guidelines for achieving high performance, such as requesting sufficiently large storage allocations so that data is striped over many BB nodes [40]. However, it does not show when, why and how to use DataWarp for specific workflow use cases. Our work has analyzed two data analytics workflows

and identified I/O signatures along with the specific workflow requirements to advise how to use DataWarp. Ovsyannikov et. al implement in-transit processing on the NERSC BB to analyze data generated by a simulation science application [41]. We did not experiment with in-transit processing because the applications implementing the different stages of SWarp and CAMP workflow pipelines cannot execute in parallel because of file dependencies. We do explain that DataWarp persistent reservations are a useful feature when the same data files and Python modules are read by multiple workflow pipelines.

7. Conclusions

In this paper, we have evaluated two important scientific workflows run at NERSC to understand their performance characteristics. We have determined their I/O characteristics, the I/O characteristics of the BB, and measured their performance when running on the BB.

Our results show that it is essential to understand the limiting performance characteristic of the scientific workflow being considered in order to achieve good performance on a BB. In this case, the applications in the two different workflows were limited by two different performance parameters: bandwidth (SWarp) and metadata (CAMP). This understanding had to be coupled with careful matching of the demands of the applications in the workflow with the BB characteristics to achieve optimal performance. Our work showed that an I/O speed-up of approximately 5× could be achieved for the CAMP workflow with proper consideration of the BB configuration requested.

These results point to the need for a richer way of enabling workflows and HPC resource allocation management software to interact in the future. Enabling a workflow to communicate its performance limiting characteristic, bandwidth, IOPS or metadata will be essential to ensure overall throughput is not decreased by inefficient usage of resources.

Acknowledgments

This work was supported by Laboratory Directed Research and Development (LDRD) funding from Berkeley Lab, provided by the Director, Office of Science and Office of Science, Office of Advanced Scientific Computing Research (ASCR) of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. The authors would also like to thank Rollin Thomas for help with installing the CAMP Python software environment on DataWarp.

References

[1] C. S. Daley, D. Ghoshal, G. K. Lockwood, S. Dosanjh, L. Ramakrishnan, N. J. Wright, Performance characterization of scientific workflows for the optimal use of burst buffers, in: Workflows in Support of Large-Scale Science (WORKS-2016), Vol. 1800, CEUR-WS.org, 2016, pp. 69–73.

[2] S. Byna, A. Uselton, D. Knaak, Y. H. He, Lessons Learned from a Hero I/O Run on Hopper, in: 2013 Cray User Group Meeting, Napa, CA, 2013.

[3] C. S. Daley, L. Ramakrishnan, S. Dosanjh, N. J. Wright, Analyses of Scientific Workflows for Effective Use of Future Architectures, in: Proceedings of the 6th International Workshop on Big Data Analytics: Challenges, and Opportunities (BDAC-15), Austin, TX, 2015.

[4] K. A. Standish, T. M. Carland, G. K. Lockwood, W. Pfeiffer, M. Tatineni, C. C. Huang, S. Lamberth, Y. Cherkas, C. Brodmerkel, E. Jaeger, L. Smith, G. Rajagopal, M. E. Curran, N. J. Schork, Group-based variant calling leveraging next-generation supercomputing for large-scale whole-genome sequencing studies, BMC Bioinformatics 16 (1) (2015) 304. doi:10.1186/s12859-015-0736-4.

URL http://dx.doi.org/10.1186/s12859-015-0736-4http://www.biomedcentral.com/1471-2105/16/304

[5] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, T. Ludwig, Small-file access in parallel file systems, in: 2009 IEEE International Symposium on Parallel & Distributed Processing, IEEE, 2009, pp. 1–11. doi:10.1109/IPDPS.2009.5161029.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5161029

[6] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, 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, pp. 1–11. doi:10.1109/MSST.2012.6232369.

[7] Trinity / NERSC-8 Use Case Scenarios, Tech. Rep. SAND 2013-2941 P, Los Alamos National Laboratory, Sandia National Laboratories, NERSC, https://www.nersc.gov/assets/Trinity--NERSC-8-RFP/Documents/trinity-NERSC8-use-case-v1.2a.pdf; accessed 4 October 2016 (Apr. 2013).

[8] K. Harms, H. S. Oral, S. Atchley, S. S. Vazhkudai, Impact of burst buffer architectures on application portability, Tech. rep., Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States). Oak Ridge Leadership Computing Facility (OLCF) (2016).

[9] D. Henseler, B. Landsteiner, D. Petesch, C. Wright, N. Wright, Architecture and Design of Cray DataWarp, in: Cray User Group CUG, 2016.
URL https://cug.org/proceedings/cug2016_proceedings/includes/files/pap105.pdf

[10] XC Series DataWarp User Guide, Tech. Rep. S-2558-5204, Cray, http://docs.cray.com/PDF/XC_Series_DataWarp_User_Guide_CLE_60UP03_S-2558.pdf; accessed 23 June 2017 (Sep. 2015).

[11] IOR, https://github.com/LLNL/ior; accessed 5 September 2016.

[12] MDTest, https://github.com/MDTEST-LANL/mdtest; accessed 5 September 2016.

[13] NASA MODIS Website, http://modis.gsfc.nasa.gov/.

[14] R. E. Wolfe, D. P. Roy, E. Vermote, Modis land data storage, gridding, and compositing methodology: Level 2 grid, IEEE Transactions on Geoscience and Remote Sensing 36 (4) (1998) 1324–1338. doi:10.1109/36.701082.

[15] V. Hendrix, L. Ramakrishnan, Y. Ryu, C. van Ingen, K. R. Jackson, D. Agarwal, CAMP: Community Access MODIS Pipeline, Future Generation Computer Systems 36 (2014) 418 – 429. doi:http://dx.doi.org/10.1016/j.future.2013.09.023.
URL http://www.sciencedirect.com/science/article/pii/S0167739X13002021

[16] Strace, http://sourceforge.net/projects/strace; accessed 13 July 2016.

[17] IPM, https://github.com/nerscadmin/IPM; accessed 13 July 2016.

[18] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, K. Riley, 24/7 Characterization of petascale I/O workloads, in: 2009 IEEE International Conference on Cluster Computing and Workshops, 2009, pp. 1–10. doi:10.1109/CLUSTR.2009.5289150.

[19] J. Liu, Q. Koziol, H. Tang, F. Tessier, W. Bhimji, B. Cook, B. Austin, S. Byna, B. Thakur, G. Lockwood, J. Deslippe, Prabhat, Understanding the IO Performance Gap Between Cori KNL and Haswell, in: Cray User Group CUG, 2017.
URL https://cug.org/proceedings/protected/cug2017_proceedings/includes/files/pap154s2-file1.pdf

[20] J. Enkovaara, N. A. Romero, S. Shende, J. J. Mortensen, Gpaw - massively parallel electronic structure calculations with python-based software, Procedia Computer Science 4 (2011) 17 – 25. doi:http://dx.doi.org/10.1016/j.procs.2011.04.003.
URL http://www.sciencedirect.com/science/article/pii/S1877050911000615

[21] Atomic commit in sqlite, https://www.sqlite.org/atomiccommit.html; accessed 4 September 2016.

[22] Database speed comparison, https://www.sqlite.org/speed.html; accessed 4 September 2016.

[23] Pragma statements, https://www.sqlite.org/pragma.html; accessed 5 September 2016.

[24] L. Ramakrishnan, B. Plale, A multi-dimensional classification model for scientific workflow characteristics, in: Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science, Wands '10, ACM, New York, NY, USA, 2010, pp. 4:1–4:12. doi:10.1145/1833398.1833402.
URL http://doi.acm.org/10.1145/1833398.1833402

[25] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. H. Su, K. Vahi, Characterization of scientific workflows, in: 2008 Third Workshop on Workflows in Support of Large-Scale Science, 2008, pp. 1–10. doi:10.1109/WORKS.2008.4723958.

[26] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, M.-H. Su, Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand (2004). doi:10.1117/12.550551.
URL http://dx.doi.org/10.1117/12.550551

[27] E. Deelman, A. Chervenak, Data management challenges of data-intensive scientific workflows, in: Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on, 2008, pp. 687–692. doi:10.1109/CCGRID.2008.24.

[28] Z. Zhang, C. Wang, S. S. Vazhkudai, X. Ma, G. G. Pike, J. W. Cobb, F. Mueller, Optimizing center performance through coordinated data staging, scheduling and recovery, in: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC '07, ACM, New York, NY, USA, 2007, pp. 55:1–55:11. doi:10.1145/1362622.1362696.
URL http://doi.acm.org/10.1145/1362622.1362696

[29] H. M. Monti, A. R. Butt, S. S. Vazhkudai, On timely staging of hpc job input data, IEEE Transactions on Parallel and Distributed Systems 24 (9) (2013) 1841–1851. doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.279.

[30] S. Bharathi, A. Chervenak, Scheduling data-intensive workflows on storage constrained resources, in: Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, WORKS '09, ACM, New York, NY, USA, 2009, pp. 3:1–3:10. doi:10.1145/1645164.1645167.
URL http://doi.acm.org/10.1145/1645164.1645167

[31] D. Ghoshal, L. Ramakrishnan, MaDaTS: Managing Data on Tiered Storage for Scientific Workflows, in: Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing, HPDC '17, ACM, New York, NY, USA, 2017, pp. 41–52. doi:10.1145/3078597.3078611.
URL http://doi.acm.org/10.1145/3078597.3078611

[32] J. Bent, G. Grider, B. Kettering, A. Manzanares, M. McClelland, A. Torres, A. Torrez, Storage challenges at los alamos national lab, in: IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), 2012, pp. 1–5. doi:10.1109/MSST.2012.6232376.

[33] K. Sato, K. Mohror, A. Moody, T. Gamblin, B. R. d. Supinski, N. Maruyama, S. Matsuoka, A user-level infiniband-based file system and checkpoint strategy for burst buffers, in: Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on, 2014, pp. 21–30. doi:10.1109/CCGrid.2014.24.

[34] B. Van Essen, R. Pearce, S. Ames, M. Gokhale, On the Role of NVRAM in Data-intensive Architectures: An Evaluation, in: 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IEEE, 2012, pp. 703–714. doi:10.1109/IPDPS.2012.69.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6267871

[35] T. Wang, S. Oral, M. Pritchard, K. Vasko, W. Yu, Development of a burst buffer system for data-intensive applications, CoRR abs/1505.01765.
URL http://arxiv.org/abs/1505.01765

[36] APEX Workflows, Tech. rep., Los Alamos National Laboratory, NERSC, and Sandia National Laboratories, Los Alamos, NM (2016).

[37] S. Herbein, D. H. Ahn, D. Lipari, T. R. Scogland, M. Stearman, M. Grondona, J. Garlick, B. Springmeyer, M. Taufer, Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters, in: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing, HPDC '16, ACM, New York, NY, USA, 2016, pp. 69–80. doi:10.1145/2907294.2907316.
URL http://doi.acm.org/10.1145/2907294.2907316

[38] S. Thapaliya, P. Bangalore, J. Lofstead, K. Mohror, A. Moody, Managing I/O Interference in a Shared Burst Buffer System, in: 2016 45th International Conference on Parallel Processing (ICPP), 2016, pp. 416–425. doi:10.1109/ICPP.2016.54.

[39] W. Bhimji, et al., Accelerating Science with the NERSC Burst Buffer Early User Program, in: Cray User Group CUG, 2016.

URL https://cug.org/proceedings/cug2016_proceedings/includes/files/pap162.pdf

[40] Burst Buffer, NERSC website: http://www.nersc.gov/users/computational-systems/cori/burst-buffer/; accessed 31 August 2016.

[41] A. Ovsyannikov, M. Romanus, B. V. Straalen, G. H. Weber, D. Trebotich, Scientific workflows at datawarp-speed: Accelerated data-intensive science using nersc's burst buffer, in: 2016 1st Joint International Workshop on Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS), 2016, pp. 1–6. doi:10.1109/PDSW-DISCS.2016.005.

Christopher S. Daley is a performance engineer in the National Energy Research Scientific Computing (NERSC) center at Lawrence Berkeley National Laboratory (LBNL). Before joining NERSC, Christopher was a Scientific Programmer at the Flash Center for Computational Science at the University of Chicago. His current research involves performance analysis of HPC and data analytics applications to gather architectural requirements and identify code optimization opportunities. He has a M.Sc. in High Performance Computing from the University of Edinburgh and a B.Sc. in Physics from the University of Surrey.

Devarshi Ghoshal is a Research Scientist at LBNL. He received his Ph.D. in Computer Science from Indiana University, Bloomington in 2014. His current research interests include high performance computing, large scale data management in distributed systems, I/O performance benchmarking and performance optimizations in scientific workflows.

Glenn K. Lockwood is a performance engineer in the NERSC center at LBNL. He specializes in I/O performance analysis, extreme-scale storage architectures, and emerging I/O technologies and APIs. His research interests revolve around understanding I/O performance by correlating performance analysis across all levels of the I/O subsystem, from node-local page cache to back-end storage devices. To this end, he is actively involved in the performance analysis of the burst buffer incorporated in Cori, NERSC's 12,000-node Cray XC-40 system, as well as the Lustre file systems deployed at the center.

Sudip Dosanjh is the director of the NERSC center at LBNL. Previously, Dr. Dosanjh headed extreme-scale computing at Sandia National Laboratories. He was co-director of the Los Alamos/Sandia Alliance for Computing at the Extreme-Scale from 2008-2012. He also served on the U.S. Department of Energy's Exascale Initiative Steering Committee for several years. Dr. Dosanjh had a key role in establishing co-design as a methodology for reaching exascale computing. He earned his bachelors degree in engineering physics in 1982, his masters degree (1984) and Ph.D. (1986) in mechanical engineering, all from the University of California, Berkeley.

Lavanya Ramakrishnan is a staff scientist at LBNL. Her research interests are in software tools for computational and data-intensive science. Ramakrishnan has previously worked as a research staff member at Renaissance Computing Institute and MCNC in North Carolina. She has masters and doctoral degrees in Computer Science from Indiana University and a bachelor degree in computer engineering from VJTI, University of Mumbai. She joined LBNL as an Alvarez Postdoctoral Fellow in 2009.

Nicholas J. Wright focuses on evaluating future technologies for potential application in scientific computing. He also works on performance measurement and optimization and is particularly involved in investigating performance optimization for the multicore-era. Before moving to NERSC, he was a member of the Performance Modeling and Characterization (PMaC) group at the San Diego Supercomputing Center. He earned both his undergraduate and doctoral degrees in chemistry at the University of Durham in England.