LLNL-CONF-831041

Towards a workload-aware storage stack for HPC systems.

H. Devarajan, K. Mohror

January 21, 2022

The ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)
Minneapolis, MN, United States
June 27, 2022 through July 1, 2022

Extracting and characterizing I/O behavior of HPC workloads

Hariharan Devarajan, Kathryn Mohror
Lawrence Livermore National Laboratory
{hariharandev1, kathryn}@llnl.gov

Abstract*—System administrators set default storage-system configuration parameters with the goal of providing high performance for their system's I/O workloads. However, this generalized configuration can lead to suboptimal I/O performance for individual workloads. Users can provide parameter settings to the storage system to obtain better performance for individual applications, but it can be very challenging to determine which parameters to set and to what values. This problem is further exacerbated by the increased complexity of modern storage systems. In this work, we move towards solving this problem by providing a systematic categorization of workload-related information that users or middleware libraries can pass to the storage system to optimize I/O performance for specific workloads. We study applications and workflows from different scientific domains to cover a broad range of HPC use cases. Through our categorization, we find that a) workload features differ based on the hardware, software, and data components involved in the execution of workloads and b) multiple workload features together drive I/O optimizations. The methodology proposed in this work optimizes complex scientific workloads by 2.2×-8×, using workload-aware I/O optimizations. Using the proposed methodology, users can pragmatically characterize their workload, and this characterization can assist the storage system in configuring itself to optimize I/O performance for individual workloads in HPC systems.

***Index Terms*—I/O characterization, workflow I/O, Montage, Simulation, LBANN, Workload-aware, Pegasus**

I. Introduction

Modern HPC workloads that generate, process, and store large amounts of data [1] are known as data-intensive workloads. These workloads consist of individual applications [2], such as simulation, analysis, and artificial-intelligence (AI) applications, and collections of applications that execute cooperatively in a workflow [3], [4], such as simulations coupled with big data or AI analytics. The diversity of HPC workloads has led to the high-configurability of storage software to cater to the different requirements of individual workloads [5]. Examples of these configurations are the Lustre file system, which allows setting the stripe size and -count parameters for configuring data distribution [6], and the HDF5 I/O library, which can be configured to optimize I/O operation granularity based on the application's access pattern, using a technique called "chunking" [7]. Typically, these configuration parameters are set to empirically-evaluated default values (e.g., the Lustre stripe size is 1MB and stripe count is 4, and the HDF5 chunk cache size is 4KB) to provide reasonable I/O performance for target workloads on an HPC system.

Default storage-system configurations are available for domain scientists who are not I/O experts [8], but using these defaultsmay lead to poor performance. For instance, a study at the Argonne Leadership Computing Facility (ALCF) revealed that 90% of workloads on Mira do not use the storage system efficiently and achieve a low aggregate bandwidth that is 1% of peak bandwidth [8]. Storage-system configuration parameters significantly affect the underlying I/O performance of the storage system. As an example, a stripe size of 1 MB for workloads that access data in the 256 KB range would reduce bandwidth achieved from the parallel file system by 2.2× [9]. Another example is that applying data compressions on data sets with certain data distributions can increase data size by 12% and increase overall time (compression + I/O time) by 1.5× [10]. Therefore, default configurations can often lead to nonoptimal I/O performance for workloads.

While setting workload-specific configurations for optimizing I/O can yield significant performance benefits, determining which parameters to set and to what values is a complex task. First of all, the workload features that should be examined to determine storage-system configurations are not well defined in the literature, so application users typically need help from I/O experts to identify the critical features of the target workload. Second, once the workload features have been identified, figuring out the configuration settings requires additional help from I/O experts and, even then, the settings are typically specific to the target storage system and may require trial and error to find the right settings. For example, I/O buffering middleware [11], [12], [13] typically allows configuration of the size of the buffer used. The buffer size can be derived from multiple workload features such as node memory, number of cores, transfer sizes, and dataset size. Due to the large number of factors that can affect the determination of the optimal buffer size, finding the right setting can be a Herculean task for non-I/O experts and challenging even for experts.

We need to move away from users' defining storage configurations because of the burdensome challenges in setting the correct parameters for I/O performance. Our vision is a new paradigm where users need only describe their workload's I/O behavior and the storage system automatically configures itself based on this description. There are two steps in reaching this goal. First, we need a methodology for systematic characterization of the I/O behavior of user workloads. Second, we need to enhance existing storage systems such that they can perform workload-specific optimizations when provided with characterization information. In this work, we focus on the first step towards our vision and develop a pragmatic methodology to characterize, extract, and map workload features that the user can pass into storage systems for performing workload-aware optimizations. We demonstrate the benefits of our approach by performing a characterization study of exemplar HPC workloads and show how the storage system can use characterization information to improve performance, citing two case studies. We characterize exemplar workloads from geology, fluid dynamics, cosmology, high-energy physics, astrophysics, and medicine and represent common I/O behaviors, including scientific simulations, checkpoint-restart, scientific AI workloads, and image processing, as well as complex workflows. We use conclusions from an existing I/O-behavior study from the literature [9], [14], [15], [16], [17], [18] and our own collected I/O profiles of the workloads to build a workload-feature characterization that holistically defines common I/O behaviors. These features are then automatically extracted using the Vani tool suite [19], a collection of tools and libraries that contains system-, job-, and workload-level information. We categorize the features into our workload-feature characterization and, finally, present the mapping of workload features into storage configurations that the storage system itself can set to optimize I/O performance. We demonstrate the effectiveness of our approach with two use cases of complex scientific workloads. The contributions of this work are

1) A methodology for systematic characterization of workload features that represent the I/O behavior in HPC workloads
2) A characterization of the I/O features of exemplar HPC workloads and mapping to storage parameter settings
3) Two case studies demonstrating how our approach identifies workload features and maps them to storage optimizations by the storage system to improve I/O time

II. Background

In this section, we discuss the complexities of achieving I/O performance on modern HPC systems, which stem from the high configurability of the software stack at multiple levels.

A. Modern storage-stack hierarchy

The diversity of HPC workloads has led to multiple software abstractions on modern storage systems, to support different I/O needs [13]. We currently have abstractions at three levels of the software stack: workload, middleware libraries, and storage software. These different levels, each with multiple storage interfaces and library options, make the configuration of the HPC storage stack a complex process for users.

At the workload level [16], abstractions provide different logical representations to meet different I/O requirements. At this level, we have user-facing high-level I/O libraries, generic I/O libraries, and lower-level primitives to access different storage devices. For instance, scientists have proposed several high-level libraries such as HDF5 [20], pNetCDF [21], and ADIOS [22]. Similarly, more generic, lower-level I/O libraries include MPI-IO [23] and STDIO [24]. Finally, at the kernel level we have primitives defined using the POSIX interface [25] or key-value store interfaces such as DAOS [26] and CephFS [27].

At the middleware level, researchers have proposed software that transparently accelerates I/O using optimizations such as buffering [12], [28], [11], prefetching [29], [30], [31], and compression [10], [32], [33]. Middleware libraries intercept I/O calls and use storage accelerators such as node-local and shared burst buffers to optimize I/O operations.

Finally, the storage layer at the lowest level is the final resting place for all data accessed across projects in an HPC system. In this level, we use long-term and highly-reliable storage devices such as raid-HDD, parallelized using parallel file systems such as Lustre [34], GPFS [35], DAOS [26], and BeeGFS [36].

B. Configurability of HPC storage stack

The storage software stack provides user-level configurations to support diverse workloads and often directly affects the performance of workloads on HPC systems [9].

In the case of parallel file systems (PFSs), each software solution provides a configuration based on its architecture. Some example configurations are as follows: a) GPFS [35], which allows users to configure the ROMIO driver to optimize for independent data accesses (locking) and enable collective I/O; b) the Intel MPI-IO library, which can be configured for an application using `cb_nodes` for setting a number of aggregators to collect small I/O into bigger buffers and a `cb_config_list` option to limit the number of processes used for collective buffering [37], c) Datawarp [38], [39] burst buffers as deployed in the Cori supercomputer to provide configurations for users to disable persistence, enable privacy of data, and define buffer capacity; and d) the DAOS [40], [26] storage system by Intel, which enables users to configure multiple tiers of storage and page-cache size.

In the case of middleware I/O libraries, each library provides optimization-specific parameters for users to optimize I/O for their workload. Some examples of these configurations are a) hierarchical buffering softwares [10], [29], [41], [13], [42] to configure parameters such as the buffer size of tiered buffering resources, placement policy, element eviction policies, etc.; b) UnifyFS [12], which allows users to set cleanup strategies, workload-specific consistency models, and synchronization options for maximizing performance; and c) Univistor [43] to enable users to configure FSTYPE for different PFS, the capacity of each buffering tier, and distribution policies for data.

Tuning configurations of system software and middleware libraries can improve I/O performance by orders of magnitude [9]. Therefore, we need the ability to extract workload features and configure the storage system to maximize I/O performance in HPC systems.

III. Methodology for I/O Characterization

Our vision is a new paradigm for I/O, where users are no longer burdened with the challenges of configuring storage systems for high performance. Instead, they need only specify the features of their workload at a high level and storage systems will automatically determine the best configurations for the workload. Currently, users have to instrument their application and manually-correlated instrument data to optimizations. In this paper, we describe the first step towards characterizing workload features. Our approach requires that we broaden our scope for I/O understanding beyond traditional scientific simulations and extract and classify the features that define modern HPC workloads. In this section, we describe our methodology, tools, and terminology and demonstrate it with exemplar HPC workloads.

A. Hardware and Software

1) Testbed: We run workloads on the Lassen supercomputer at Lawrence Livermore National Laboratory (LLNL) [44]. It is a 23-petaflop IBM Power9 system and the twenty-sixth supercomputer in the world (as of November 2021), based on the TOP500 list [45]. Lassen has 795 nodes connected with a Mellanox 100 Gb/s enhanced data rate (EDR) InfiniBand network and a 24 PiB IBM Spectrum Scale file system (also known as GPFS). Individual Lassen nodes consist of two IBM POWER9 CPUs (IBM AC922 servers) with 256GB of system memory and four Nvidia Volta V100 GPUs with 64GB of HBM2 memory. The Lassen supercomputer is a smaller and unclassified version of LLNL's larger Sierra system [46], the third-fastest supercomputer in the world, based on the TOP 500 list [45]. The Sierra supercomputer has a peak performance of 125 petaflops, with 4,474 nodes.

2) Tools: We chose Recorder [47] to gather comphrensive traces of the workloads because Recorder is the only tracing tool that provides multilevel I/O traces along with CPU and GPU calls. We use multicomponent traces (I/O, CPU, and GPU) to perform complex data-dependency analysis on applications and workflows. For instance, the I/O behavior depends not only on the I/O trace of the application, but also on the use of computing elements such as CPU and GPU, owing to the potential use of overlapped and parallel I/O [9]. Capturing detailed information adds overhead to workload runtime. This study notes an overhead of 8% on the workload runtime, due to trace collection by the Recorder. Additionally, we use the Vani tool suite [19] and extend it to build analysis scripts to extract workload features from Recorder traces. All the results described in this paper are the outcome of the analysis produced by the Vani tool suite with our extensions (available at https://bit.ly/3OXolnm). The features extracted by the Vani suite are used to manually reconfigure I/O for storage systems.

3) Terminology: In this work, a *workload* expresses a job running on an HPC system. This job could execute a single application or a multi-application workflow. For the characterization, we define an *Entity* that expresses the hardware, software, and data components of the system. The entity abstraction enables us to group several workload features that belong to the same component in the system. We specifically focus on I/O-related entities. Finally, we define *Attributes* as features of an entity.

B. Exemplar HPC Workloads

To extend the understanding of modern HPC workloads, we need a group of representative workloads from different scientific domains. The workloads should encompass diverse I/O behaviors in modern HPC systems that are not included in existing characterization work in the literature, such as scientific AI applications and complex multistage workflows. For completeness, we include detailed understanding of popular scientific simulations, checkpoint-restart, and image-processing workloads. These include CM1, an atmospheric-simulation model [48], HACC-I/O, an I/O kernel for hardware/hybrid accelerated cosmology [49], CosmoFlow, a deep-learning application for cosmological simulation [50], the JAG ICF model [51], and two workflows using the Montage mosaic engine (one with MPI [52], the other with Pegasus [53], [52]). These applications and workflows are large-scale HPC workloads that we execute over 32 nodes to understand their I/O behavior.

1) CM1: is an atmospheric-simulation workload that models phenomena such as thunderstorms and tornadoes. It simulates a part of the atmosphere as a fixed 3D array in which each point is characterized by a set of variables such as temperature, pressure, and windspeed. The simulation occurs over 193 steps to generate different pieces of the atmosphere. The application primarily generates data using a set of configuration files. These files are 16MB in size, and the model generates more than 750 files for different simulation steps. Each step generates files with a total size of around 128MB.

2) HACC: is a cosmology workload that simulates the universe's evolution using particle-mesh techniques. HACC-IO is an isolated I/O kernel of the workload, released within the CORAL benchmark suite and representing a typical I/O workload as found in scientific simulations. The application runs with 16M particles as input, and each process writes nine variables. The benchmark writes the simulation data as a checkpoint and emulates restart by reading the checkpointing data back. We select the File-Per-Process version (using POSIX) of the benchmark as shared MPI-IO reads are represented in the CosmoFlow application. The benchmark generates and reads files per process. Every process reads and writes 632MB; the total data generated is 790GB.

3) CosmoFlow: estimates the values of critical cosmological parameters from a 3D cosmological simulation using deep learning. A singleton application, it uses AI, rather than simulation, to constrain the effects of systematic falls, which are computationally-expensive simulations. Creating surrogates for these simulations is necessary to generate the staggering statistical numbers needed to control the systematics, and therefore it is part of the MLPerf-HPC benchmark. The CosmoFlow dataset labeled "2019_05_4parE" contains 10K simulated universes with four redshifts (channels) and 5123 voxels, stored as 16-bit integers, with the four cosmological parameters used by the workload to generate the universe. The 1.5TB dataset contains 50K samples of 32MB each, stored as an HDF5 file.

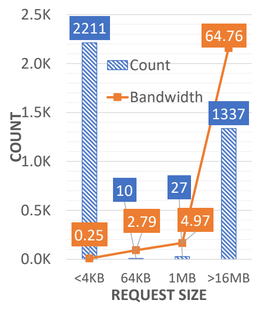
4) JAG ICF: is a semi-analytic AI model of ICF implosions in 3D. This singleton application uses AI instead of simulation to evolve an ICF capsule through the final stages of a NIF experiment. The application produces scalar, time-series, and hyperspectral ray-traced images of the implosion, which are directly compared to experiments. The dataset is formatted as a 200MB NumPy array (npy) consisting of 100K samples. The application consumes this dataset with a batch size of 128 samples over hundred epochs and a learning rate of 0.0001.

5) Montage using MPI: is a mosaic engine that converts sky-survey data formatted as Flexible Image Transport System (FITS) files into a PNG image for survey NGC 3372. In this version, the workflow consists of six stages with a data-parallel mosaic building. The collection of FITS images is divided into multiple segments, where a compute-node processes each one. Different segments are processed in parallel and each segment of the sky creates a PNG image of the survey. 1024 FITS files were divided among 32 nodes, with each node having one segment of sky containing 16 FITS files. The workflow consists of three logical steps per node, that is, sequential, parallel, and sequential jobs.

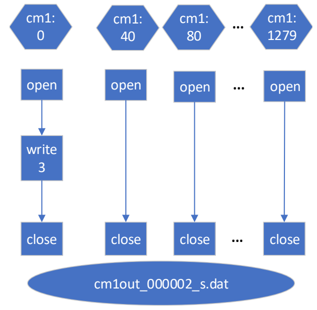
6) Montage using Pegasus: This workflow uses the same mosaic engine as our previous workload, but instead of simply converting sky-survey data into a png, it transforms all the images in the surveys to a common pixel scale of 1 second or arc, where all the pixels are co-registered on the sky and represented in galactic coordinates and Cartesian projection. The output of this workflow covers the 10° of the sky along the galactic plane where each mosaic image is 5° by 5° and has an overlap of 1° with the neighboring tiles. The workflow uses nine kernels of the mosaic engine in a complex workflow to build these patches of sky. The workflow is executed over 32 nodes using the Pegasus workflow manager [54] with pegasus-mpi-cluster [55]. The pegasus-mpi-cluster schedules these nine kernels over 1280 mpi processes to execute a distributed and parallel workflow job.

C. Generalization of characterization

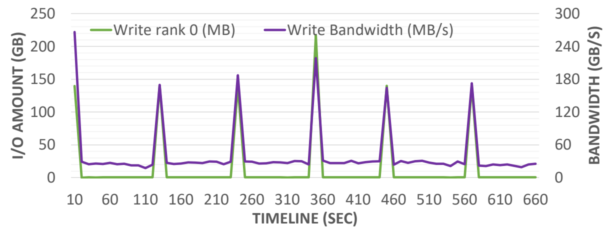
The characterization of I/O behavior is inspired not only by our "I/O behavior for workloads", as described in the next section, but also on I/O characterization performed in the literature [9], [14], [15], [16], [17], [18], [2], [56], [57], [58], [59]. The two main reasons for performing I/O analysis in this work are a) to extend the applications represented with new HPC workloads such as AI using LBANN and complex workflows using Pegasus and b) to provide the level of analysis needed to perform the characterization required by new tracing tools such as Recorder [47] (as opposed to Darshan [58]). Recorder provides more detailed I/O tracing, as decribed in the tools section of the characterization. In addition to using a diverse set of characterizations from the literature, we also diversify our HPC system configurations based on modern supercomputers such as Cori [60], Summit [61], Aurora [62], and El Capitan [63]. In the systems context, we include concepts such as node-local burst buffers, shared burst buffers, and heterogeneous CPU–GPU architectures, which are present in the different architectures of these systems. For better generalization of the ideas presented in this work, we also provide a collection of tools to extract the characterized features automatically from the tracing provided by Recorder.



(a) Request Size and Bandwidth hist.



(b) Process and Data Dependency.



(c) Write timeline of CM1.

Fig. 1: I/O behavior of CM1: Figure a) shows that large reads achieve 64GB/s aggregate bandwidth compared to small writes, which achieve 64MB/s. Figure b) shows the file (80MB) opened by many processes, but only rank 0 writes to it. Figure c) shows that rank 0 writing the simulation data achieves low bandwidth (95% of I/O time).

These features can be loaded by any storage system and perform automatic configurations for optimizing I/O.

IV. I/O Characterization for Workloads

In this section, we present the results of our characterization methodology on our exemplar workloads.

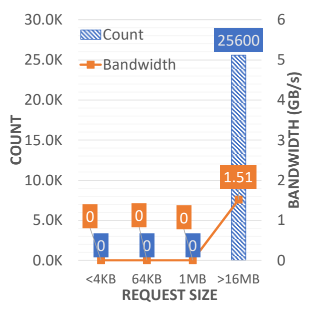
A. I/O behavior for workloads

A high-level summary of workloads is presented in Table I, followed by a detailed analysis of I/O behavior.

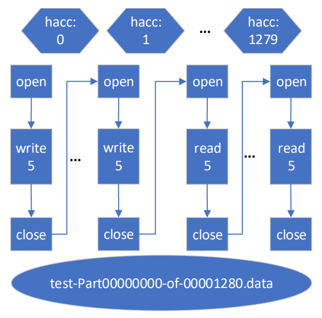
TABLE I: High-Level I/O behavior of applications.

I/O Behavior	CM1	HACC (FPP)	Cosmoflow (HDF5)	JAG	Montage MPI	Montage Pegasus
job time (sec)	664	33	3567	1289	247	1038
% of I/O time	11%	75%	12%	13%	12%	21%
Write I/O	1GB	750GB	20MB	2MB	24GB	32GB
Read I/O	20GB	750GB	1.5TB	25GB	28GB	106GB
CPU Cores/node	40	40	40	40	1-40	1-40
# files used	774	1280	50K	1	1040	5738
Shared File access	37	0	50K	1	80	960
File per process (FPP) access	737	1280	0	0	960	4778
Access Pattern	Sequential					
I/O Interface	POSIX	POSIX	HDF5-MPI-IO	STDIO	STDIO	STDIO

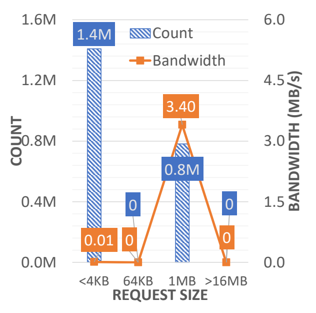
1) CM1 [Figure 1]: In this application, 20GB is for reading the 16MB configuration files by all ranks and 1GB is for writing simulation data by rank 0 [Figure 1(a)]. For the writes, every first rank per node (i.e., 40, 80,...,1240) opens and closes the file, but only rank 0 writes the simulation data [Figure 1(b)]. The application has separate read, write, and compute phases. The application initially reads the configuration



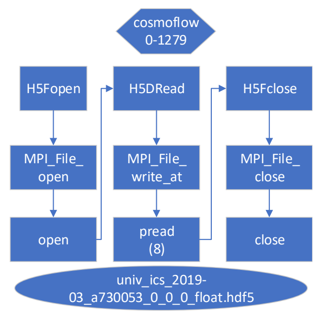
(a) Request Size and Bandwidth hist.



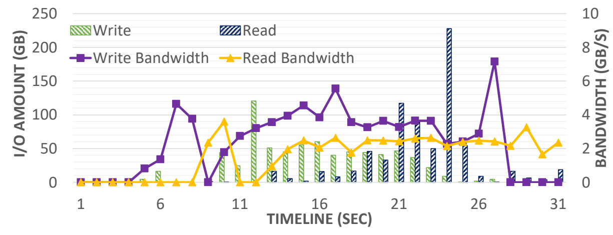
(b) Process and Data Dependency.



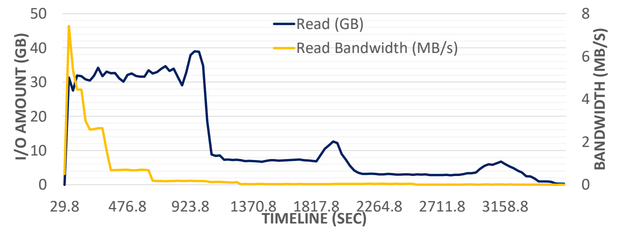
(a) Request Size and Bandwidth hist.



(b) Process and Data Dependency.



(c) I/O timeline of HACC.



(c) Read timeline of Cosmoflow.

Fig. 2: I/O behavior of HACC: Figure a) shows that large 16MB reads achieve 1.5GB/s bandwidth/process. Figure b) shows the checkpoint file (632MB) written and read by a single process, with multiple opens and closes. Figure c) shows all ranks achieve a high bandwidth of 7GB/s for writing and 3.8GB/s for reading checkpoint data.

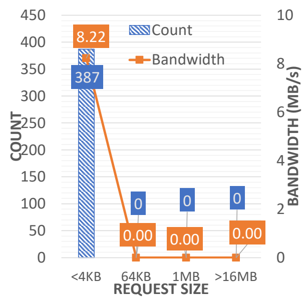
Fig. 3: I/O behavior of Cosmoflow: Figure a) shows that small accesses achieve a low bandwidth of 100KB/s and 1MB data reads achieve 3.5MB/s aggregate bandwidth. Figure b) shows the input file is read by all processes using HDF5 with MPI-IO. Figure c) shows all ranks achieve a low bandwidth of 3.5MB/s for reading simulation input data.

files (20GB in 3 sec in Figure 1(c)) and then performs computation and writing of simulation data (1GB in 10 sec) alternating [Figure 1(a)]. The application presents the simulation data in a 3D representation in memory and writes it to the file with transfer sizes of 4KB in a sequential-access pattern. The small transfer size issued for the writes by rank 0 dominates time spent in I/O with a low aggregate bandwidth of 64MB/s. Additionally, 87.5% of the time is spent in metadata operations.

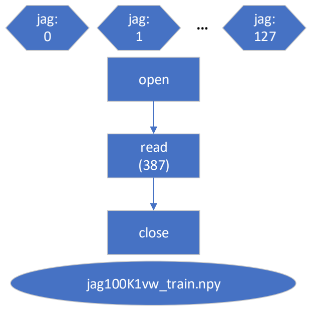
2) HACC [Figure 2]: In this workload, 50% I/O time is spent on metadata operations (4× more metadata operations than read or write operations). This large percentage of metadata operations is unexpected, as the application performs large sequential I/O [Figure 2(a)]. Upon investigating the process/data dependency, we observe that the files are opened and closed multiple times for each checkpoint and restart process. The application first generates the variables in memory then writes 750GB of data using ten write operations, and finally reads back the 750GB data as a part of restart [Figure 2(b)]. The simulation data used in the checkpoint is presented as single-dimensional variables, and each write is performed in 16MB granularity in a sequential pattern. Even with a consistent I/O access pattern, each rank achieves a different GPFS bandwidth [Figure 2(c)]. This inconsistency in bandwidth is due to contention for parallel file-system resources, owing to the high parallelism of requests occurring during checkpointing.

3) CosmoFlow [Figure 3]: The application performs I/O on the CPU while the computation runs on the GPU. The application uses four processes per node to allocate GPU for the deep-learning (DL) computations and uses 40 CPU cores for performing I/O. Each process gets a low I/O bandwidth [Figure 3(a)]. The application achieves a maximum aggregated read bandwidth of 7.4MB/s [Figure 3(c)]. This low bandwidth results from the collective MPI–IO accesses on 32MB files with 1MB transfer size per process, which hurt the overall performance owing to the number of files accessed [Figure 3(b)]. There is no file chunking (i.e., the file is represented as a one big chunk of 1D bytes instead of chunks of data) in HDF5 files [7], which slows down the multiple metadata accesses (4× more than read operation) on the dataset, due to collective I/O. This is apparent in that 98% of the I/O time is spent in metadata ops and only 2% in data ops. The application writes 20MB checkpoint files with small 40K operations periodically during DL computations. Both reads and writes on the data occur sequentially, without any seek operations.

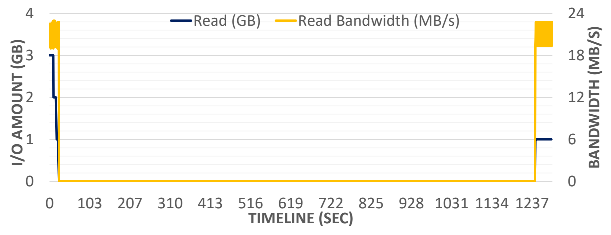
4) JAG ICF [Figure 4]: performs I/O using the STDIO interface used by NumPy array Python files. The application performs computations on GPU while executing the input pipeline on the CPU. Every rank reads 2MB's worth of samples (a portion of the complete 200MB dataset) during the first epoch and then caches the result for later epochs [Figure 4(c)]. The application performs validation at the end of the application, which is the second I/O phase at the end. Each epoch writes 20KB of checkpoint files to GPFS. Each access is less than 4KB transfer size, due to the small samples in the dataset [Figure 4(a)]. Additionally, 70% of the operations are metadata



(a) Request Size and Bandwidth hist.

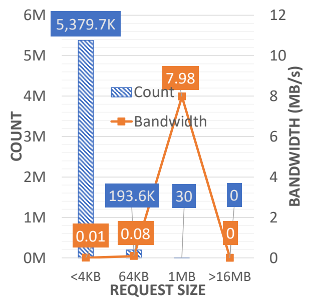


(b) Process and Data Dependency.

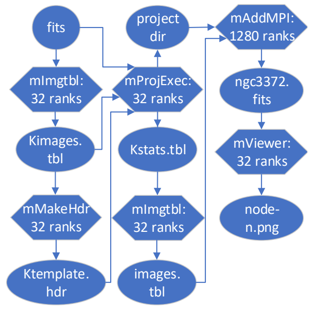


(c) Read timeline of JAG ICF.

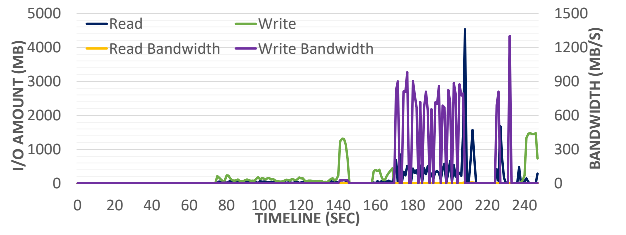
Fig. 4: I/O behavior of JAG ICF: Figure a) shows that JAG performs many small accesses with a low bandwidth of 8MB/s. Figure b) shows the input file is read by all processes using the Numpy API. Figure c) shows all ranks achieve a low bandwidth of 20MB/s for reading input data at start and end.



(a) Request Size and Bandwidth Hist.



(b) App and Data Dependency.



(c) I/O timeline of Montage.

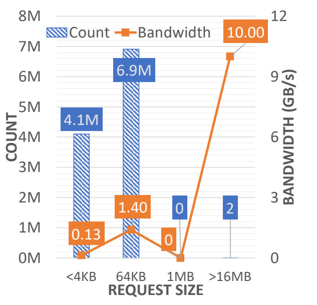
Fig. 5: I/O behavior of Montage using MPI: Figure a) shows that a lot of small access are made with a low bandwidth of 0.1MB/s. Figure b) shows the multistage execution of workflow. Figure c) shows all ranks achieve a average/request bandwidth of 5MB/s for read and 91MB/s for write ops.

operations (80% of job time); every rank opens and closes the file once and reads multiple chunks of samples from the file.

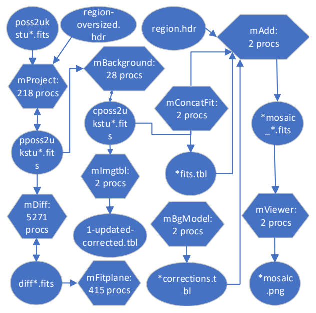
5) Montage using MPI [Figure 5]: The first rank on every node performs 40× more I/O than the rest of the processes because of the sequential parts in every node of the workflow. Of six stages, only one uses a parallel job to process the projected files [Figure 5(b)]. In the workflow, 960 files are input FITS images, whereas 80 files are intermediate files produced and consumed by the workflow. The workflow performs 21GB I/O on the input FITS files. However, the I/O time is primarily dominated by data operations on the files, with a small number of metadata operations [Figure 5(b)]. The workflow performs 4M read operations and 1M write operations. The data is represented as a 1D array in memory and written/read sequentially using less than 4KB transfer size (90%) for intermediate files. The input files are read with 64KB transfer size, which constitutes the other 10% of operations [Figure 5(a)]. However, the bandwidth distribution throughout the execution of the program varies widely [Figure 5(c)]. The average bandwidth per request for read and write operations is low (5MB/s and 91MB/s, respectively), but the application for certain specific durations (<20%) has a high I/O bandwidth of 600–1300MB/s for writes and 400–1200MB/s for reads. This spike occurs because of some buffering effects of the client nodes where data was written and immediately read by the next process.

The workflow consists of five applications that produce data which is consumed by the next step. The workflow generates small metadata files with extensions .hdr and .tlb. The mAddMPI is a parallel job using MPI, running with 1280 processes on 32 nodes. The remaining executables are executed in parallel using a shell script with one process per node. The distribution of I/O operations per application shows that the majority of read/write operations in the workflow is from the mAddMPI and mViewer (2.45M and 1.1M, respectively) which is 1000× more than other processes. The bulk of I/O performed is by mAddMPI (3.7GB reads and 25GB writes) and mViewer (24GB reads and 115MB writes), which consititutes 98% of I/O operations.

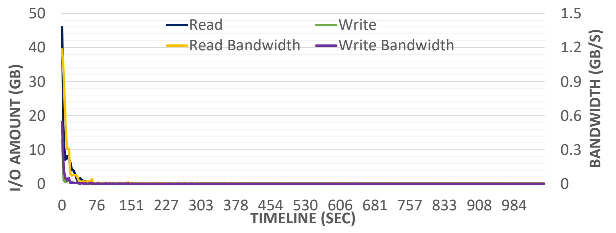
6) Montage using Pegasus [Figure 6]: performs 138GB of I/O, of which 60% is performed by mDiff reading data, 12% by mBackground for reads and writes, and 10% by mProject writing data. In the workflow, 4778 files are initial-input files read by mProject, mConcatFit, and mBgModel to generate the next step of data and metadata files for the workflow [Figure 6(b)]. The other 960 files are intermediate files produced and consumed during the workflow. In the first 20 seconds of the workflow, 80GB of I/O is performed by the mDiff and mProject applications. The workflow subsequently performs 400MB–7GB I/O by mProject and mBackground for the next 150 seconds and finally 67–114MB by mFirPlane, mConcatFit, mViewer, mImgTbl, mAdd, and mBgModel for the rest of the time [Figure 6(c)]. The workflow creates and accesses data with a small 64KB or less transfer size. This results in a small aggregate bandwidth of 140MB/s for <4KB and 1.3GB/s for 64KB transfer sizes. The mViewer performs two large requests of >16MB, which get a high I/O bandwidth of 10GB/s [Figure 6(a)]. The nine kernels are executed by the pegasus-mpi-cluster and spawn 6039 processes (of which



(a) Request Size and Bandwidth Hist.



(b) App and Data Dependency.



(c) I/O timeline of Montage.

Fig. 6: I/O behavior of Montage with Pegasus: Figure a) shows small accesses achieve a low bandwidth of 130MB/s for 4KB and 1.4GB/s for 64KB. Figure b) shows the data dependency of various executables with the workflow. Figure c) shows the workflow perform more I/O initially and thus get better I/O bandwidth.

5209 processes are of mDiff) that consume and produce data for building the mosiac engine. The largest data generated was by mViewer, generating mosaic images of 1.5GB. However, the workflow spends more time on smaller files, as data is accessed using smaller transfer sizes of <4KB. Finally, across the kernels of the workflow, 65% of the time was spent on read/write operations and 35% on metadata operations.

B. Systematic characterization of I/O in workloads

Based on our investigation of different features of the workloads, above and in the literature, we aim to build a characterization of modern HPC workloads using high-level groups of entities. Each characterization in this section is developed for a single workload running on a specific instance of software and storage stack. In the categorization, the workload uses a particular collection of software, a collection of allocated hardware resources, and uses/produces a collection of data. At the highest level of abstraction, we propose three main types of entities for a given instance of a workload: a) job entity type, b) software entity type, and c) data entity type. These entities encapsulate different aspects of a job executed in a HPC system, such as job allocation from the scheduler, the software stack available to execute for the job, including various storage systems, and the data itself that the job produces/consumes.

The *Job* entity type describes the various aspects related to job submission of the workload. These aspects include a) job-configuration entity type (Table II): attributes of a workload corresponding to job scheduling and allocation of resources;

TABLE II: Attributes for Job Configuration Entity Type.

Attribute	CM1	HACC	Cosmoflow	JAG	Montage MPI	Montage Pegasus
# nodes	32					
# cpu cores per node	40					
#gpu/node	4					
# Node-local BB dir	/dev/shm					
# Shared BB dir	NA					
PFS dir	/p/gpfs1					
Job time	2hr	2hr	6hr	6hr	2hr	12hr

TABLE III: Attributes for Job Configuration Entity Type.

Attribute	CM1	HACC	Cosmoflow	JAG	Montage MPI	Montage Pegasus
# CPU cores used/node	40	40	4	4	40	40
# GPUs used/node	0	0	4	4	0	0
# apps	1	1	1	1	5	5
App data dependency	NA	NA	NA	NA	Fig 5(b)	Fig 6(b)
FPP/shared file access	737/ 37	1280/ 0	0/ 49664	0/ 2	960/ 80	4778/ 960
I/O amount	21GB	1.5TB	1.5TB	200MB	53GB	139GB
I/O ops dist (data, meta)	30%, 70%	50%, 50%	2%, 98%	30%, 70%	99%, 1%	65%, 35%
Runtime (sec)	664	33	3567	1289	247	1038

TABLE IV: Attributes for Application Entity Type.

Attribute	CM1	HACC	Cosmoflow	JAG	Montage MPI	Montage Pegasus
# processes	1280	1280	128	128	1280	2
Process data depedency	Fig 1(b)	Fig 2(b)	Fig 3(b)	Fig 4(b)		
FPP/shared file access	737/ 37	1280/ 0	0/ 49664	0/ 2	850/ 0	1028/ 0
I/O amount	21GB	1.5TB	1.5TB	200MB	53GB	139GB
I/O ops dist (data, meta)	30%, 70%	50%, 50%	2%, 98%	30%, 70%	99%, 1%	85%, 15%
Interface	POSIX	POSIX	HDF5	STDIO	STDIO	STDIO
Runtime	664sec	33sec	3567sec	1289sec	112sec	43sec

TABLE V: Attributes for I/O Phase Entity Type. First Phase.

Attribute	CM1	HACC	Cosmoflow	JAG	Montage MPI	Montage Pegasus
I/O amount	20GB	725GB	1.5TB	200MB	1.5GB	2.08GB
I/O ops dist (data, meta)	99%/ 1%	100%/ 0%	2%, 98%	30%, 70%	99%/ 1%	99%/ 1%
Frequency	1 op	7 ops/ rank	Iterative (1MB)	Iterative (4KB)	Bulk (64KB)	Bulk (64KB)
Runtime	0.3sec	18.3sec	392sec	167sec	0.3sec	10.24sec

b) workflow entity type (Table III): attributes of a workload corresponding to workflow behavior and interactions at a high level; c) application entity type (Table IV): attributes of a workload corresponding to an individual application in the workload and the relationships among its processes; and d) I/O-phase entity type (Table V): attributes of a workload corresponding to each I/O phase within an application defined using a threshold between two I/O calls. These aspects describe the different scopes of job entities to capture attributes at different workload levels.

The *Software* entity type encompasses the different software layers of a modern HPC storage system. These layers include a) high-level I/O entity type (Table VI): attributes of a workload that describe the features of high-level libraries used within the workload; b) middleware-libraries type (Table VII): attributes of the workload that define the behavior of each

TABLE VI: Attributes for High-Level I/O Entity Type.

Attribute	CM1	HACC	Cosmoflow	JAG	Montage MPI	Montage Pegasus
Data repr	3D	1D	3D	3D	4D	2D
Granularity (data, meta)	4KB-16MB	16MB	1MB 4KB	4KB	64KB	64KB
Access pattern	Seq	Seq	Seq	Seq	Seq	Seq
Data dist	normal	uniform	gamma	normal	uniform	uniform

TABLE VII: Attributes for Middleware I/O Entity Type. As the workload used no middleware library, I/O granularity and access pattern do not change.

Attribute	CM1	HACC	Cosmoflow	JAG	Montage MPI	Montage Pegasus
# extra cores for I/O/node	0	0	36	36	0-39	0-39
Granularity (data, meta)	4KB-16MB	16MB	1MB 4KB	4KB	64KB	64KB
Memory/ node	128GB	200GB	196GB	200GB	196GB	128GB
Access pattern	Seq	Seq	Seq	Seq	Seq	Seq

TABLE VIII: Attributes for Node-Local Storage Entity Type.

Attribute	CM1	HACC	Cosmoflow	JAG	Montage MPI	Montage Pegasus
# parallel ops (controller)	64					
Capacity /node	128GB	200GB	196GB	200GB	196GB	128GB
Max I/O bw/node	32GB/s					
Dir	/tmp					

TABLE IX: Attributes for Shared-Storage Entity Type.

Attribute	CM1	HACC	Cosmoflow	JAG	Montage MPI	Montage Pegasus
# parallel servers	>2000					
Capacity /node	20PB					
Max I/O BW	64GB/s using 32 node IOR					
Dir	/p/gpfs1					

middleware library used directly or indirectly (through interception) within the workload; and c) storage-system entity type, such as the node-local storage-system entity type (Table VIII) and shared storage-system entity type (Table IX): attributes that define the characteristics of the storage system, including hardware and driver-level information accessible by the user of the workload. These layers cover all the complex layers of the storage software stack in modern HPC systems.

Finally, the *Data* entity groups the attributes that describe workload data. This entity is further divided into two entity types: a) dataset (Table X) and b) file (Table XI). The attributes of the dataset entity describe the complete data produced/accessed by the workload at a high level. The attributes of the file entity correspond to individual files created/accessed by the workload. These attributes expand on different properties of the dataset in storage systems.

C. Automatic workload characterization

The attributes described in previous subsections can be extracted using the Vani tool Suite. The suite consists of the JobUtility and Analyzer tools, which extract information from

TABLE X: Attributes for Dataset Entity Type.

Attribute	CM1	HACC	Cosmoflow	JAG	Montage MPI	Montage Pegasus
Format	bin	bin	HDF5	bin	bin	bin
Size (GB)	20.32	750	1.5TB	0.2	30	37
# of files	774	1280	49664	2	1040	5738
I/O (GB)	21	1.5TB	1.5TB	0.2	53	139
Time (sec)	73	24	392	167	29	218
I/O ops dist (data, meta)	30%, 70%	50%, 50%	2%, 98%	30%, 70%	99%, 1%	99%, 1%
File size dist (data,config)	1GB 16MB	160MB	32MB	200MB	20GB	37GB

TABLE XI: Attributes for File Entity Type. Data File

Attribute	CM1	HACC	Cosmoflow	JAG	Montage MPI	Montage Pegasus
Format	bin	bin	HDF5	bin	bin	bin
Size (GB)	0.1	0.6	0.03	0.200	0.03	3.8
I/O (GB)	0.1	1.2	0.03	0.2	0.03	3.8
Time (sec)	12	24	0.007	167	0.02	3.28
I/O ops dist (data, meta)	30%, 70%	50%, 50%	2%, 98%	30%, 70%	99%, 1%	99%, 1%
Format attributes	NA	NA	- chunk:NA - #datasets: 1 - #dims: 3	- type: float - #datasets: 1 - #dims: 3	- type: int - #dims: 3 - enc:FITS	- type: int - #dims: 2 - enc:FITS

the workloads and then build the workload attributes. We attach a Recorder profiler [47] to trace the application for CPU function calls, GPU function calls, MPI communication calls, and multilevel I/O information during the workload execution. After execution, we run our `JobUtility` tool to extract job-level information such as system-specific information (e.g., the total number of nodes and # cpu and GPU cores allocated per node) and data-specific information (e.g., number and format of files in the dataset and size of each file). The Recorder and utility logs are collected and stored in a global file syste such as GPFS. After job execution, we use the `Analyzer` tool to convert the Recorder logs into parquet format. This is a necessary first step, as Recorder logs are in row-major format and filtering and aggregation operations in memory are highly inefficient for this format. The parquet file stores data in column-major format, which can be processed in an out-of-core fashion using the DASK tool [64]. The DASK library is used within the Analyzer to extract application information and generate a YAML file of entities and attributes with workload-specific values. We illustrate the working of our `Analyzer` using Jupyter notebooks (available at https://bit.ly/3OXolnm).

D. Optimizing workloads based on characterization

After characterizing different aspects of the workload as in the previous section, let's explore the potential optimizations that storage systems can achieve with this user-provided information from the workloads. Note that this list is incomplete and depends on the system, available middleware libraries, and the nature of the workloads. We provide these guidelines to drive more workload-aware optimization in HPC systems.

1) I/O acceleration through software techniques: The literature demonstrates that accelerating I/O using software techniques such as aggregation [65], buffering [13], caching [66], compression [10], and prefetching [29], [30] is extremely common within modern HPC systems. These optimizations may use attributes across all entities of a workload. The attributes that can be used based on the study of the optimizations mentioned above are *# nodes*, to identify the scale of application and deploy I/O services appropriately; *# cpu cores/node*, to decide the scale of I/O services per node; *# gpu/node*, to use GPU for accelerating data operations such as compression; *# of apps in workflow & # of processes in app*lication, to identify various apps within the workload for scheduling I/O; *data dependency for apps & processes*, to optimize the data path of the workload; *I/O amount*, to identify the critical data path in the workload; *I/O Phase Frequency*, to handle critical phases of the workload with more resources; *I/O granularity*, to determine cache line size and buffer granularities in algorithms; *I/O access pattern*, to tune eviction policies and buffering strategies to match workload pattern; *% of memory used*, for configuring buffering and caching resources; *# parallel operations supported*, to identify parallelism of operations for ideal I/O; *# parallel I/O servers*, to identify parallelism of operations for ideal I/O; and *distribution of I/O per file*, to identify important files in a workload.

2) Async I/O Optimization: To perform asynchronous I/O inherently for workloads, storage systems and middleware libraries require workload attributes to enable these optimizations and still maintain correctness [12]. Some examples of these attributes are *# nodes in job*, to maintain coherency flags; *node-local/shared BB dir*, to use BB as a buffering resource for data; *I/O amount and ops per file*. to identify important files; *I/O phase frequency, and runtime bound*, to identify overlapping I/O with compute; and *data dependency between processes*, to identify synchronization points for the workload.

3) Optimize software systems for workloads: Users and middleware libraries can use this workload attribute to optimize storage systems such as PFS and burst buffers. For instance, we can optimize the Lustre filesystem by setting the stripe size [34] using *# I/O ops in workload per file* to identify the important files and *I/O granularity per operation* to identify the transfer size for important files. The transfer size selected would be set to stripe size to optimize the most important accesses. Another example is GPFS, where users can set the lock in ROMIO driver [35] to false if there is no *data dependency in apps and processes*. In Datawarp burst buffers, users can disable data persistency using the `DisablePersistent` flag. This flag can be set based on *data dependency between apps* and *I/O granularity* to understand whether any data needs to be stored at the end or all data is temporary.

4) Process placement for workflow emulators: Workflow emulators are often required to place various applications and processes in the workflow on specific nodes to maximize data locality and reduce I/O cost [3]. To achieve this, we use attributes such as *data dependency between apps & processes* to identify data locality, *# nodes* to set the scale for the workflow, *# cpu cores/node & # gpus/node* to identify available resources, and *dirs for node-local and shared BB* to optimize workflow I/O using accelerators.

5) Optimize dataset: Users can optimize their workloads by improving the layout and attributes of the dataset they consume [9], [10]. To enable these optimizations, we use attributes from the workload such as *format of the dataset* to enable format-specific optimizations such as chunking and compression in HDF5, *size of the dataset* to enable storage-level preloading or caching, *# samples* to understand the granularity of I/O, *I/O amount & time spent* to identify important files, and *distribution of I/O operations* to identify which operations are most important, such as data or metadata.

The attributes for each optimization are not exhaustive but are to be used as a guideline to make the HPC storage system workload-aware.

V. Use cases

Based on the characterization of entities and attributes for modern HPC workloads, we can define entities and attributes for each workload and explore potential optimizations the storage system can perform to optimize workloads. To illustrate the benefit of this characterization, we use the workloads Cosmoflow and Montage as use cases to identify the attributes (shown in Section IV-B) with which users can optimize the workloads on the Lassen supercomputer (described in Section III) at LLNL. In all cases, we demonstrate the automatic characterization of workload through the Vani tool suite, storage reconfiguration for optimizing workload's I/O, and the impact of these reconfigurations on the workload's runtime. Currently, reconfiguration is performed manually, but we envision storage-system software can automatically perform these optimizations based on the attributes of the workload provided.

A. Cosmoflow workload

1) Automated characterization: We extract the features of the workload using the Vani tool suite, with results as follows: We observe that workload consumed 49K files each of size 32MB (1.5TB total size) in a shared, accessed pattern across all processes. Due to the small file size and the MPI–IO driver in HDF5, we see many metadata calls in the applications. As a result, 98% of I/O time is spent on metadata calls. This behavior is further supported by the distribution of I/O operations in the workloads read (390K), write (38K), and metadata (1.3M). All data was accessed sequentially across the workload, with 391K read operations for 1.5TB data.

2) Optimizing workload through storage reconfiguration: Based on the workload features extracted, we need to minimize the cost of metadata access in HDF5 files. To achieve this, we preload the data initially into the shared memory of the application, as 196GB of memory is not utilized in each node. We can fit 1/32 of the dataset on each shared memory and perform the read directly from there. The use of shared memory would reduce the cost of I/O and limit the aggregation of files using MPI–IO to a node. We expect this optimization to reduce the two bottlenecks we identified using workload attributes (i.e., metadata access of the HDF5 file and the aggregation of small files across many processes). To perform the preloading, we use MPIFileUtils to load dataset pieces in parallel into each node's shared memory within the job and configure LBANN to read the dataset from the shared memory location.

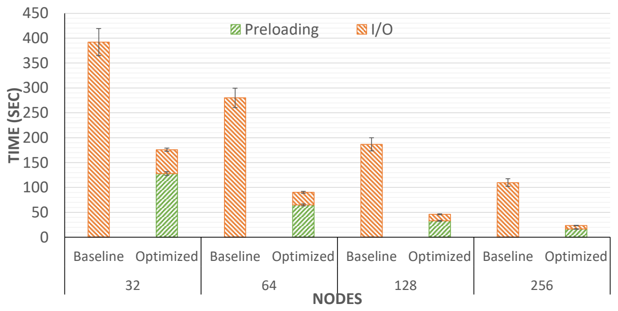


Fig. 7: Optimizing Cosmoflow using workload attributes demonstrates that GPFS cannot support highly concurrent MPI-IO accesses on small 32MB files, due to the high cost for metadata and I/O interference. We preload the data into shared memory and reduce parallel access to files to optimize this behavior and improve I/O performance up to 4.6×

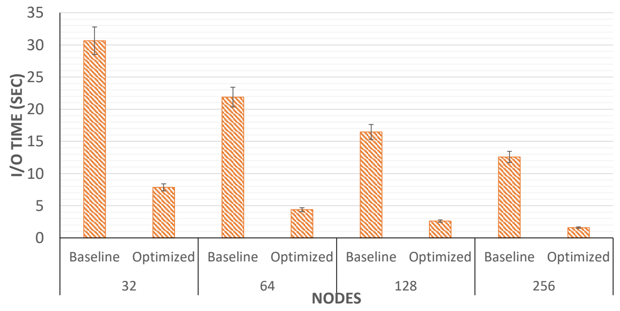


Fig. 8: Optimizing Montage using workload attributes demonstrates that a large number of intermediate files accessed through GPFS are not optimal, due to small transfer sizes on these files. We use shared memory for these accesses and improve I/O up to 8× on larger scales.

3) Experimental results: For testing the benefits of reconfiguration, we strong scale the workload from 32–256 nodes in powers of two [Figure 7]. For the baseline (B in the figure), we run the Cosmoflow app over GPFS. As the scale increases, we observe a decrease in I/O time. However, this improvement is sublinear to ideal scaling (i.e., 1.25×-1.4×), due to the metadata access on HDF5 on GPFS and interference of MPI–IO on small HDF5 files on the PFS. For the optimized version (O in the figure), we preload the data using an MPI job (using MPIFileUtils) into shared memory and then run Cosmoflow over that dataset. Here the data preloading scales linearly with an increase in scale as $\#files >> \#nodes$. Additionally, the I/O in this optimized version scales almost linearly, due to the optimized local access to HDF5 metadata and node-level sharing of files using MPI–IO. The preloading of data into shared memory increases the overall performance of the workload by 2.2×-4.6×.

B. Montage with MPI workload

1) Automated characterization: The attributes of the workflow are presented as follows. We observe that 1024 input fits files were consumed by the initial step, accounting for 1.5Gb of I/O, and that the final png file generated is 5MB in size. However, the workload generated and consumed 53GB of data. This large I/O size is due to the generation and subsequent consumption of intermediate files, which the workflow need not store on the GPFS. We see the majority of operations performed were pure data operations (i.e., 99%), and these operations are skewed towards reads rather than writes (i.e., 4M reads vs. 1M writes). Also, 90% of the operation are small I/O from the access of intermediate files. The I/O on intermediate files accounts for 95% of I/O time. Finally, intermediate files are generated and accessed locally to processes in a node.

2) Optimizing workload through storage reconfiguration: Based on the extracted workload attributes, we need to accelerate I/O for intermediate files, which would optimize 95% of I/O time. We can store and access these intermediate files using a node-local burst buffer to achieve this. To emulate a node-local burst buffer in Lassen, we use shared memory, as the application uses only a small amount of memory for its execution. Given the size of generated intermediate files (800MB per node), we can easily fit all the intermediate files on node-local shared memory and optimize small accesses. To perform this optimization, we use the application parameters to change the directory location where the files are produced/accessed in the job script and perform optimized I/O.

3) Experimental results: For testing these optimizations, we strong scale the workload on 256 nodes [Figure 8]. For the baseline (B in the figure), we run Montage workflow over GPFS. As the scale increases, we observe that I/O time decreases. However, I/O improves sublinearly (i.e., 1.35×-1.5×), as 95% of the I/O time is still dependent on intermediate files, accessed using small transfer sizes on the GPFS. For the optimized version (O in the figure), all the steps of the workload store intermediate files on shared memory and read it for the consumer steps. As all stages depend on previous stages, the intermediate files created are consumed by the next step and the small accesses are optimized through local shared memory. The use of shared memory for intermediate files optimizes the I/O significantly, with an overall improvement in workload of 3.9×-8×.

VI. Related Work

A. Understand, characterize, and optimize I/O behavior

The process of optimizing I/O workloads has been developed extensively over the past decade. The process involves understanding I/O behavior [67] through profiling and analysis, characterizing I/O behavior using clustering and generalization, and finally, optimizing workloads to match the designs of the storage system. To facilitate this process, scientists have developed several tools that acquire information from workloads, analyze the data, and optimize the workload based on this analysis. The primary target of these tools is to extract I/O information from interfaces such as STDIO [24], POSIX [25], and MPI-IO [23], or higher-level I/O libraries such as HDF5 [20], pNetCDF [21], and ADIOS [22]. For acquisition, we have a plethora of profiling [16], [68], [69] and tracing tools [47], [67], [16], [70] to extract different levels of detail from the workload. For analysis, these tools form companion tools to existing acquisition tools [67], [71], [19], [72]. However, their goal is to improve I/O behavior themselves. By contrast, our work focuses on making the storage system workload-aware and transparently optimizing diverse workloads.

B. *Workload-aware I/O optimizations*

In response to the diversity of workloads, storage systems have begun providing mechanisms for workload awareness. Examples include a) buffering software [13], [11], [12] that requires users to configure allocated buffering resources, data placement policies, etc.; b) prefetching software [30], [29], [31] that uses the data-access pattern information collected during runtime to build prefetching schemes; c) compression libararies [10], [73] that use data distribution, type, and format to apply the most appropriate compression library at runtime; d) high-performance object stores [26], [74] that require users to configure sharding policies and replication servers for objects to match workload behavior; and e) PFSs [35], [34] that require users to configure stripe size, stripe count, and collective I/O to match the workload's data-access pattern and data distribution. These software solutions demonstrate the effectiveness of workload information for maximizing I/O performance for HPC workloads. As shown, however, each software solution employs its feature set to optimize I/O. Our work agrees with these solutions to build workload-aware storage and moreover advocates the standardization of workload features for easy adoption in HPC environments.

VII. Conclusion

Modern scientific workloads contain several related and unrelated features that can assist storage systems in optimizing I/O for workloads. In this work, we characterized workflow features into entities and attributes that can be utilized by storage systems to perform workload-aware I/O optimizations. Our investigation of workloads and systems has uncovered three key workload features in modern HPC systems. First, workload features differ based on hardware, software, and data components involved in the execution of the workload. Second, workload features need to be extracted at different stages of the workload, such as offline, job allocation, and job execution. Finally, the multiple workloads featured together drive I/O optimizations through storage configurations. We demonstrate the effectiveness of workflow-aware I/O optimizations on complex scientific workloads such as Montage and CosmoFlow (gaining a 2.2x–8x speedup in I/O performance) and provide a systematic approach to extracting workload features and optimizing I/O performance for individual workloads in HPC systems. We plan to use these attributes to enhance storage software stacks with workload-aware optimizations.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract LLNL-CONF-831041. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under the DOE Early Career Research Program.

References

[1] M. Mercier, D. Glesser, Y. Georgiou, and O. Richard, "Big data and HPC collocation: Using HPC idle resources for Big Data analytics," in *2017 IEEE International Conference on Big Data (Big Data)*, Dec. 2017, pp. 347–352.

[2] H. Luu, B. Behzad, R. Aydt, and M. Winslett, "A multi-level approach for understanding I/O activity in HPC applications," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, Sep. 2013, pp. 1–5, iSSN: 2168-9253.

[3] F. Chowdhury, Y. Zhu, F. Di Natale, A. Moody, E. Gonsiorowski, K. Mohror, and W. Yu, "Emulating I/O Behavior in Scientific Workflows on High Performance Computing Systems," in *2020 IEEE/ACM Fifth International Parallel Data Systems Workshop (PDSW)*, Nov. 2020, pp. 34–39.

[4] S. D. Mohaghegh, "Reservoir simulation and modeling based on artificial intelligence and data mining (AI&DM)," *Journal of Natural Gas Science and Engineering*, vol. 3, no. 6, pp. 697–705, Dec. 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1875510011001090

[5] B. Behzad, H. V. T. Luu, J. Huchette, S. Byna, Prabhat, R. Aydt, Q. Koziol, and M. Snir, "Taming parallel I/O complexity with auto-tuning," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. Denver Colorado: ACM, Nov. 2013, pp. 1–12. [Online]. Available: https://dl.acm.org/doi/10.1145/2503210.2503278

[6] U. T. A. U. o. N. HPC group, "Lustre FS performance tips — HPC documentation 0.0 documentation." [Online]. Available: https://hpc-uit.readthedocs.io/en/latest/storage/lustre-performance.html

[7] P. Elena and K. Larry, "HDF5 Data Compression Demystified #2: Performance Tuning," May 2017. [Online]. Available: https://www.hdfgroup.org/2017/05/hdf5-data-compression-demystified-2-performance-tuning/

[8] H. Luu, M. Winslett, W. Gropp, R. Ross, P. Carns, K. Harms, M. Prabhat, S. Byna, and Y. Yao, "A Multiplatform Study of I/O Behavior on Petascale Supercomputers," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '15. New York, NY, USA: Association for Computing Machinery, Jun. 2015, pp. 33–44. [Online]. Available: https://doi.org/10.1145/2749246.2749269

[9] H. Devarajan, H. Zheng, A. Kougkas, X.-H. Sun, and V. Vishwanath, "DLIO: A Data-Centric Benchmark for Scientific Deep Learning Applications," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, May 2021, pp. 81–91.

[10] H. Devarajan, A. Kougkas, L. Logan, and X.-H. Sun, "HCompress: Hierarchical Data Compression for Multi-Tiered Storage Environments," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2020, pp. 557–566, iSSN: 1530-2075.

[11] H. Tang, S. Byna, F. Tessier, T. Wang, B. Dong, J. Mu, Q. Koziol, J. Soumagne, V. Vishwanath, J. Liu, and R. Warren, "Toward Scalable and Asynchronous Object-Centric Data Management for HPC," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2018, pp. 113–122.

[12] A. Moody, D. Sikich, N. Bass, M. J. Brim, C. Stanavige, H. Sim, J. Moore, T. Hutter, S. Boehm, K. Mohror, D. Ivanov, T. Wang, and C. P. Steffen, "UnifyFS: A Distributed Burst Buffer File System - 0.1.0," Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States), Tech. Rep. UnifyFS, Oct. 2017. [Online]. Available: https://www.osti.gov/biblio/1408515

[13] A. Kougkas, H. Devarajan, and X.-H. Sun, "Hermes: a heterogeneous-aware multi-tiered distributed I/O buffering system," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '18. New York, NY, USA: Association for Computing Machinery, Jun. 2018, pp. 219–230. [Online]. Available: https://doi.org/10.1145/3208040.3208059

[14] M. Agarwal, D. Singhvi, P. Malakar, and S. Byna, "Active Learning-based Automatic Tuning and Prediction of Parallel I/O Performance," in *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*, Nov. 2019, pp. 20–29.

[15] Y.-T. S. Chang, H. Jin, and J. Bauer, "Methodology and Application of HPC: I/O Characterization with MPIProf and IOT," in *2016 5th*

Workshop on Extreme-Scale Programming Tools (ESPT), Nov. 2016, pp. 1–8.

[16] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular HPC I/O Characterization with Darshan," in *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, Nov. 2016, pp. 9–17.

[17] F. Isaila, P. Balaprakash, S. M. Wild, D. Kimpe, R. Latham, R. Ross, and P. Hovland, "Collective I/O Tuning Using Analytical and Machine Learning Models," in *2015 IEEE International Conference on Cluster Computing*, Sep. 2015, pp. 128–137, iSSN: 2168-9253.

[18] S. Snyder, P. Carns, R. Latham, M. Mubarak, R. Ross, C. Carothers, B. Behzad, H. V. T. Luu, S. Byna, and Prabhat, "Techniques for modeling large-scale HPC I/O workloads," in *Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*, ser. PMBS '15. New York, NY, USA: Association for Computing Machinery, Nov. 2015, pp. 1–11. [Online]. Available: https://doi.org/10.1145/2832087.2832091

[19] H. Devarajan, "VaniDL," Dec. 2021, original-date: 2020-06-13T02:14:38Z. [Online]. Available: https://github.com/hariharan-devarajan/vanidl

[20] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the HDF5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, ser. AD '11. New York, NY, USA: Association for Computing Machinery, Mar. 2011, pp. 36–47. [Online]. Available: https://doi.org/10.1145/1966895.1966900

[21] J. Li, W.-k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, "Parallel netCDF: A High-Performance Scientific I/O Interface," in *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, Nov. 2003, pp. 39–39.

[22] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin, "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)," in *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, ser. CLADE '08. New York, NY, USA: Association for Computing Machinery, Jun. 2008, pp. 15–24. [Online]. Available: https://doi.org/10.1145/1383529.1383533

[23] R. Thakur, E. Lusk, and W. Gropp, "Users guide for ROMIO: A high-performance, portable MPI-IO implementation," Argonne National Lab. (ANL), Argonne, IL (United States), Tech. Rep. ANL/MCS-TM-234, Oct. 1997. [Online]. Available: https://www.osti.gov/biblio/564273

[24] M. R. M. Dunsmuir and G. J. Davies, "Buffered Input and Output," in *Programming the UNIX™ System*, ser. Macmillan Computer Science Series, M. R. M. Dunsmuir and G. J. Davies, Eds. London: Macmillan Education UK, 1985, pp. 80–95. [Online]. Available: https://doi.org/10.1007/978-1-349-07371-9_5

[25] B. Gallmeister, *POSIX.4 Programmers Guide: Programming for the Real World*. "O'Reilly Media, Inc.", 1995, google-Books-ID: 4Kb_1sKprCMC.

[26] M. S. Breitenfeld, N. Fortner, J. Henderson, J. Soumagne, M. Chaarawi, J. Lombardi, and Q. Koziol, "DAOS for Extreme-scale Systems in Scientific Applications," *arXiv:1712.00423 [cs]*, Dec. 2017, arXiv: 1712.00423. [Online]. Available: http://arxiv.org/abs/1712.00423

[27] P. Llopis, C. Lindqvist, N. Høimyr, D. v. d. Ster, and P. Ganz, "Integrating HPC into an agile and cloud-focused environment at CERN," *EPJ Web of Conferences*, vol. 214, p. 07025, 2019, publisher: EDP Sciences. [Online]. Available: https://www.epj-conferences.org/articles/epjconf/abs/2019/19/epjconf_chep2018_07025/epjconf_chep2018_07025.html

[28] Y. Chen, M. Winslett, S.-w. Kuo, Y. Cho, M. Subramaniam, and K. Seamons, "Performance modeling for the panda array I/O library," in *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM) - Supercomputing '96*. Pittsburgh, Pennsylvania, United States: ACM Press, 1996, pp. 45–es. [Online]. Available: http://portal.acm.org/citation.cfm?doid=369028.369122

[29] H. Devarajan, A. Kougkas, and X.-H. Sun, "HFetch: Hierarchical Data Prefetching for Scientific Workflows in Multi-Tiered Storage Environments," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2020, pp. 62–72, iSSN: 1530-2075.

[30] P. Subedi, P. Davis, S. Duan, S. Klasky, H. Kolla, and M. Parashar, "Stacker: An Autonomic Data Movement Engine for Extreme-Scale Data Staging-Based In-Situ Workflows," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2018, pp. 920–930.

[31] J. He, X.-H. Sun, and R. Thakur, "KNOWAC: I/O Prefetch via Accumulated Knowledge," in *2012 IEEE International Conference on Cluster Computing*, Sep. 2012, pp. 429–437, iSSN: 2168-9253.

[32] B. Welton, D. Kimpe, J. Cope, C. M. Patrick, K. Iskra, and R. Ross, "Improving I/O Forwarding Throughput with Data Compression," in *2011 IEEE International Conference on Cluster Computing*, Sep. 2011, pp. 438–445, iSSN: 2168-9253.

[33] T. Makatos, Y. Klonatos, M. Marazakis, M. D. Flouris, and A. Bilas, "Using transparent compression to improve SSD-based I/O caches," in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys '10. New York, NY, USA: Association for Computing Machinery, Apr. 2010, pp. 1–14. [Online]. Available: https://doi.org/10.1145/1755913.1755915

[34] P. Braam, "The Lustre Storage Architecture," *arXiv:1903.01955 [cs]*, Mar. 2019, arXiv: 1903.01955. [Online]. Available: http://arxiv.org/abs/1903.01955

[35] F. Schmuck and R. Haskin, "{GPFS}: A Shared-Disk File System for Large Computing Clusters," 2002. [Online]. Available: https://www.usenix.org/conference/fast-02/gpfs-shared-disk-file-system-large-computing-clusters

[36] J. Heichler, "Introduction to BeeGFS," p. 11.

[37] "ANL/MCS-TM-234 Users Guide for ROMIO: A High-Performance, Portable MPI-IO Implementation," 2007.

[38] "Slurm Workload Manager - Slurm Burst Buffer Guide." [Online]. Available: https://slurm.schedmd.com/burst_buffer.html#command-line-dw

[39] "Slurm Workload Manager - burst_buffer.conf." [Online]. Available: https://slurm.schedmd.com/burst_buffer.conf.html

[40] "Changing the DAOS tier 2 configuration." [Online]. Available: https://help.hcltechsw.com/domino/earlyaccess/admn_daos_t2_config_changes.html

[41] H. Devarajan, A. Kougkas, and X.-H. Sun, "HReplica: A Dynamic Data Replication Engine with Adaptive Compression for Multi-Tiered Storage," in *2020 IEEE International Conference on Big Data (Big Data)*, Dec. 2020, pp. 256–265.

[42] A. Kougkas, H. Devarajan, J. Lofstead, and X.-H. Sun, "LABIOS: A Distributed Label-Based I/O System," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '19. New York, NY, USA: Association for Computing Machinery, Jun. 2019, pp. 13–24. [Online]. Available: https://doi.org/10.1145/3307681.3325405

[43] T. Wang, S. Byna, B. Dong, and H. Tang, "UniviStor: Integrated Hierarchical and Distributed Storage for HPC," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, Sep. 2018, pp. 134–144, iSSN: 2168-9253.

[44] "Lassen | High Performance Computing." [Online]. Available: https://hpc.llnl.gov/hardware/platforms/lassen

[45] TOP500.org, "November 2021 | TOP500," Nov. 2021. [Online]. Available: https://www.top500.org/lists/top500/2021/11/

[46] "Sierra | High Performance Computing." [Online]. Available: https://hpc.llnl.gov/hardware/platforms/sierra

[47] C. Wang, J. Sun, M. Snir, K. Mohror, and E. Gonsiorowski, "Recorder 2.0: Efficient Parallel I/O Tracing and Analysis," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2020, pp. 1–8.

[48] "Coupled surface-atmosphere reflectance (CSAR) model: 1. Model description and inversion on synthetic data - Rahman - 1993 - Journal of Geophysical Research: Atmospheres - Wiley Online Library." [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/93JD02071?casa_token=6uBtJPfrS64AAAAA:XvK2Ee-5OIjelSCq80R0ohCdLLqtC7fIjLa-rk9_Jz7DjZhNdug5JTKLqqvdtiL1Cyy328qtcn44CDvj

[49] S. Habib, V. Morozov, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, T. Peterka, J. Insley, D. Daniel, P. Fasel, N. Frontiere, and Z. Lukic, "The Universe at extreme scale: Multi-petaflop sky simulation on the BG/Q," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Nov. 2012, pp. 1–11, iSSN: 2167-4337.

[50] Y. Oyama, N. Maruyama, N. Dryden, E. McCarthy, P. Harrington, J. Balewski, S. Matsuoka, P. Nugent, and B. Van Essen, "The Case for Strong Scaling in Deep Learning: Training Large 3D CNNs With Hybrid Parallelism," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1641–1652, Jul. 2021, conference Name: IEEE Transactions on Parallel and Distributed Systems.

[51] J. L. Peterson, K. Athey, P. T. Bremer, V. Castillo, F. Di Natale, J. E. Field, D. Fox, J. Gaffney, D. Hysom, S. A. Jacobs, J. Koning, B. Kostowski, S. Langer, P. Robinson, J. Semler, B. Spears, B. Van Essen, J. S. Yeom, B. Kailkhura, and J. Thiagarajan, "Merlin: Enabling Machine Learning-Ready HPC Ensembles," Lawrence Livermore National Lab. (LLNL), Livermore, CA (United States), Tech. Rep. LLNL-CONF-771638, Apr. 2019. [Online]. Available: https://www.osti.gov/biblio/1630805

[52] M. Reagan, *The Hand of God: Thoughts and Images Reflecting the Spirit of the Universe*. Andrews McMeel Publishing, 1999, google-Books-ID: MAqtdv3H3nIC.

[53] M. Rynge, G. Juve, J. Kinney, J. Good, B. Berriman, A. Merrihew, and E. Deelman, "Producing an Infrared Multiwavelength Galactic Plane Atlas Using Montage, Pegasus, and Amazon Web Services," vol. 485, p. 211, May 2014, conference Name: Astronomical Data Analysis Software and Systems XXIII ADS Bibcode: 2014ASPC..485..211R. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2014ASPC..485..211R

[54] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. Ferreira da Silva, M. Livny, and K. Wenger, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, May 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X14002015

[55] E. Deelman, K. Vahi, M. Rynge, R. Mayani, R. F. da Silva, G. Papadimitriou, and M. Livny, "The Evolution of the Pegasus Workflow Management Software," *Computing in Science Engineering*, vol. 21, no. 4, pp. 22–36, Jul. 2019, conference Name: Computing in Science Engineering.

[56] W.-k. Liao, "Design and Evaluation of MPI File Domain Partitioning Methods under Extent-Based File Locking Protocol," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 2, pp. 260–272, Feb. 2011, conference Name: IEEE Transactions on Parallel and Distributed Systems.

[57] M. Howison, "Tuning HDF5 for Lustre File Systems," Sep. 2010. [Online]. Available: https://escholarship.org/uc/item/46r9d86r

[58] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 Characterization of petascale I/O workloads," in *2009 IEEE International Conference on Cluster Computing and Workshops*, Aug. 2009, pp. 1–10, iSSN: 2168-9253.

[59] H. Shan, K. Antypas, and J. Shalf, "Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark," in *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Nov. 2008, pp. 1–12, iSSN: 2167-4337.

[60] K. Antypas, N. Wright, N. P. Cardo, A. Andrews, and M. Cordery, "Cori: A Cray XC Pre-Exascale System for NERSC," p. 5.

[61] J. Wells, B. Bland, J. Nichols, J. Hack, F. Foertter, G. Hagen, T. Maier, M. Ashfaq, B. Messer, and S. Parete-Koon, "Announcing Supercomputer Summit," Oak Ridge National Lab. (ORNL), Oak Ridge, TN (United States), Tech. Rep., Jun. 2016. [Online]. Available: https://www.osti.gov/sciencecinema/biblio/1259664

[62] "Aurora | Argonne Leadership Computing Facility." [Online]. Available: https://www.alcf.anl.gov/aurora

[63] "Cray Wins NNSA-Livermore 'El Capitan' Exascale Contract," Aug. 2019. [Online]. Available: https://www.hpcwire.com/2019/08/13/cray-wins-nnsa-livermore-el-capitan-exascale-award/

[64] M. Dugré, V. Hayot-Sasson, and T. Glatard, "A Performance Comparison of Dask and Apache Spark for Data-Intensive Neuroimaging Pipelines," in *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, Nov. 2019, pp. 40–49.

[65] A. Ching, A. Choudhary, K. Coloma, W.-k. Liao, R. Ross, and W. Gropp, "Noncontiguous I/O accesses through MPI-IO," in *CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings.*, May 2003, pp. 104–111.

[66] V. S. Pai, P. Druschel, and W. Zwaenepoel, "IO-Lite: a unified I/O buffering and caching system," *ACM Transactions on Computer Systems*, vol. 18, no. 1, pp. 37–66, Feb. 2000. [Online]. Available: https://dl.acm.org/doi/10.1145/332799.332895

[67] H. Devarajan, A. Kougkas, P. Challa, and X.-H. Sun, "Vidya: Performing Code-Block I/O Characterization for Data Access Optimization," in *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, Dec. 2018, pp. 255–264, iSSN: 2640-0316.

[68] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, and W. E. Nagel, "The Vampir Performance Analysis Tool-Set," in *Tools for High Performance Computing*, M. Resch, R. Keller, V. Himmler, B. Krammer, and A. Schulz, Eds. Berlin, Heidelberg: Springer, 2008, pp. 139–155.

[69] S. Shende, A. D. Malony, W. Spear, and K. Schuchardt, "Characterizing I/O Performance Using the TAU Performance System," *Applications, Tools and Techniques on the Road to Exascale Computing*, pp. 647–655, 2012, publisher: IOS Press. [Online]. Available: https://ebooks.iospress.nl/doi/10.3233/978-1-61499-041-3-647

[70] A. Knüpfer, C. Rössel, D. a. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, W. E. Nagel, Y. Oleynik, P. Philippen, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf, "Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope,Scalasca, TAU, and Vampir," in *Tools for High Performance Computing 2011*, H. Brunst, M. S. Müller, W. E. Nagel, and M. M. Resch, Eds. Berlin, Heidelberg: Springer, 2012, pp. 79–91.

[71] "PyDarshan Documentation — PyDarshan 3.3.1.0 documentation." [Online]. Available: https://www.mcs.anl.gov/research/projects/darshan/docs/pydarshan/index.html

[72] C. Wang, "recorder-viz," Dec. 2021, original-date: 2021-01-27T02:29:28Z. [Online]. Available: https://github.com/wangvsa/recorder-viz

[73] H. Devarajan, A. Kougkas, and X.-H. Sun, "An Intelligent, Adaptive, and Flexible Data Compression Framework," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2019, pp. 82–91.

[74] "MongoDB in Action: Covers MongoDB version 3.0 - Kyle Banker, Douglas Garrett, Peter Bakkum, Shaun Verch - Google Books." [Online]. Available: https://books.google.com/books?hl=en&lr=&id=kzkzEAAAQBAJ&oi=fnd&pg=PT21&dq=MongoDB&ots=8U30rS2366&sig=YBW3ssfDeXbqdvw3eZUP6GVeKO4#v=onepage&q=MongoDB&f=false