Understanding HPC Application I/O Behavior Using System Level Statistics

Arnab K. Paul*, Olaf Faaland†, Adam Moody†, Elsa Gonsiorowski†,
Kathryn Mohror†, Ali R. Butt*
*Virginia Tech, †Lawrence Livermore National Laboratory
{akpaul, butta}@vt.edu, {faaland1, moody20, gonsiorowski1, mohror1}@llnl.gov

***Abstract*—The processor performance of high performance computing (HPC) systems is increasing at a much higher rate than storage performance. This imbalance leads to I/O performance bottlenecks in massively parallel HPC applications. Therefore, there is a need for improvements in storage and file system designs to meet the ever-growing I/O needs of HPC applications. Storage and file system designers require a deep understanding of how HPC application I/O behavior affects current storage system installations in order to improve them. In this work, we contribute to this understanding using application-agnostic file system statistics gathered on compute nodes as well as metadata and object storage file system servers. We analyze file system statistics of more than 4 million jobs over a period of three years on two systems at Lawrence Livermore National Laboratory that include a 15 PiB Lustre file system for storage. The results of our study add to the state-of-the-art in I/O understanding by providing insight into how general HPC workloads affect the performance of large-scale storage systems. Some key observations in our study show that reads and writes are evenly distributed across the storage system; applications which perform I/O, spread that I/O across ∼78% of the minutes of their runtime on average; less than 22% of HPC users who submit write-intensive jobs perform efficient writes to the file system; and I/O contention seriously impacts I/O performance.**

I. Introduction

The current trend for high performance computing (HPC) systems is that processor performance improves at a rate of 20% per year, while disk access time improves by only 10% every year [6]. As a result, massively parallel HPC applications can suffer from imbalance in computation and I/O performance, with I/O operations becoming a limiting factor in application efficiency [14]. To mitigate this problem, much effort has been dedicated to implementing high performance parallel file systems to support the I/O needs of HPC applications. The Lustre file system [24] is one of the most widely-used parallel file systems, supporting seven of the top ten supercomputers in the latest Top-500 list (June, 2020) [1].

Even though HPC parallel file systems, such as Lustre, provide much higher bandwidth and reliability than other options, continual improvement of parallel file systems is needed to reduce the impact of the increasing I/O performance bottleneck. File system designers need a comprehensive understanding of the I/O workloads on current HPC systems so that they can design successful next-generation file systems. Additionally, HPC system designers and system administrators need to understand both file system and application behavior so that they can design and tune HPC systems to run as efficiently as possible. Traditionally, I/O and storage analysis efforts have concentrated on analyzing application I/O behavior from application-level statistics [7], [8], [12], [17], [23], [27], [34], and there have been significant studies on the I/O workloads of large scale systems [2], [3], [4], [14], [18], [39], [40] which identify potential I/O bottlenecks in applications and suggest improvements to HPC users. However, while these studies can give clues about how particular applications utilize and stress HPC file systems, they do not give true insight into storage system performance under a general load. Thus, in order to draw meaningful conclusions about how to improve parallel file system designs for all users of an HPC system, we need to analyze statistics captured from the file system itself (from file system data on compute nodes and from data on servers that manage the I/O requests from HPC applications) independently of user applications.

Some of the earliest studies of HPC file systems utilizing system statistics were in 2010 [36], [43]. These works analyzed the deployment of Lustre file systems and lessons learned from them. However, to the best of our knowledge, there have not been efforts which analyze file system statistics in an application-agnostic manner to understand how to support a general HPC workload. Lawrence Livermore National Laboratory (LLNL) is home to a variety of clusters that utilize Lustre file systems as primary storage, and millions of jobs run on these systems. In this effort, we take advantage of the Lustre resources at LLNL and perform a study of general application behavior using file system statistics from Lustre.

In this paper, we collect and study file system statistics from two Livermore Computing systems with 15 PiB Lustre file systems at LLNL, namely `Quartz` and `Cab`[1]. We collect two types of data from these systems.

- *Aggregate Job Statistics:* This data represents aggregate statistics collected from file system daemons on compute nodes for all jobs that ran on these two systems during the logging period: for Cab April 2015 – March 2018, and for Quartz April 2017 – March 2018.
- *Time-Series Job Statistics:* This data represents time series data collected at 60-second intervals from the metadata server and object storage servers of Lustre for each job; i.e., for each job, we record summary statistics

[1]Cab was decommissioned in June 2018.

for every minute of the running job during which I/O operations occurred. We collected this data from Quartz for the period June 7, 2018 – July 10, 2018.

We analyze these system statistics without knowledge of the types of applications running on these systems with the goal of answering questions such as:

- What are the typical I/O characteristics of I/O-heavy jobs? For example, do the jobs perform efficient writes with large byte counts per operation? Or do they perform many small (inefficient) write requests?
- How do I/O operations of jobs affect the metadata server?
- Is I/O traffic heavier on any particular day of the month or day of the week?
- Can a single long-running job have a significant affect on the performance of the object storage servers?
- How does the size of application output files correlate with compute node memory size?
- How do metadata operations spread across the different metadata servers?
- Does I/O contention affect the I/O performance in the system?

Our study of application-agnostic I/O statistics from the Lustre file system contributes to the state-of-the-art in I/O behavior understanding. We provide insight into how general HPC workloads affect the performance of file systems, which can aid system architects in improving file system and storage system designs and system administrators in tuning existing systems and advising users of best practices. These improvements will help to alleviate the I/O imbalance in HPC systems and increase the overall efficiency of HPC applications.

II. Background

In this section, we first describe the architecture of the Lustre file system. Following this, we describe LLNL computing systems we utilized in this work.

A. Lustre Distributed File System

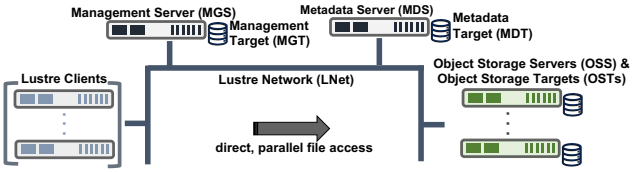


Fig. 1: An overview of Lustre architecture.

The architecture of the Lustre file system is shown in Figure 1. Lustre has a client-server network architecture and is designed for high performance and scalability. The *Management Server (MGS)* is responsible for storing the configuration information for the entire Lustre file system. This persistent information is stored on the *Management Target (MGT)*. The *Metadata Server (MDS)* manages all the namespace operations for the file system. The namespace metadata, such as directories, file names, file layout, and access permissions are stored in an *Metadata Target (MDT)*. Every Lustre file system must have a minimum of one MDT. *Object Storage Servers (OSSes)* provide the storage for the file contents in a Lustre file system. Each file is stored on one or more *Object Storage Target (OST)*s mounted on the OSS. Applications access the file system data via *Lustre clients* which interact with OSSes directly for parallel file accesses. The internal high-speed data networking protocol for Lustre file system is abstracted and is managed by the *Lustre Network (LNet)* layer.

B. Clusters

	Cab	**Quartz**
Processor Architecture	Xeon 8-core E5-2670	Xeon 18-core E5-2695
Operating System	TOSS 2	TOSS 3
Processor Clock Rate	2.6 GHz	2.1 GHz
Nodes	1,296	2,634
Cores per node	16	36
Total Cores	20,736	96,768
Memory per node	32 GB	128 GB
Total Memory	41.5 TB	344.06 TB
Interconnect	QDR Infiniband	Intel Omni-Path 100 Gb/s
Tflops	426.0	3,251.4

TABLE I: Cluster Configurations.

Table I gives an overview of the two compute clusters (`Cab` and `Quartz`) at LLNL used in our study. Both clusters have a 15 PiB Lustre file system as primary storage.

III. Data Collection

We collected two categories of file system data from `Cab` and `Quartz`.

- Aggregate Job Statistics
- Time-Series Job Statistics

A. Aggregate Job Statistics

The aggregate job statistics were collected on the client (compute) nodes, by gathering Lustre counter data just before and after the job runs, and calculating the difference. The counter data are acquired from */proc/fs/lustre/llite/lustre-file-system/stats* which is exported by the Lustre client. The statistics in this file reflect the requests as they pass through the interface between the Linux Virtual File System (VFS) and Lustre. Any file system request handled entirely by VFS is not included, nor is a background activity such as re-transmission of a low-level Lustre (not user process) request after server recovery from a crash. All the read- and write-related system calls result in VFS calling into Lustre code, so the read- and write-related counters we use reflect every system call the job made. The statistics are per file system, not per Lustre OSS. These counters start at 0 when the Lustre file system is mounted, and are monotonically increasing until they wrap at $2^{63} - 1$.

The specific statistics used are:

- *starttime, endtime, duration, uid, nodes*: These give the time when the job was started, when it ended, the duration of the job, the anonymized user ID of the job submitter, and the number of nodes on which the job ran.
- *mkdir, mknod, open, rename, rmdir, unlink*: These are the total metadata statistics recorded for the job.

- *read_bytes, write_bytes*: These represent the total number of bytes read and written by the job to the Lustre file system.
- *read_bytes_count, write_bytes_count*: These give the number of read and write calls made by the job to the Lustre file system.
- *recv_bytes, recv_count, send_bytes, send_count*: These network statistics give the number of packets received and sent as well the number of bytes received and sent over LNet.

Both `Cab` and `Quartz` use the SLURM job scheduler [35]. SLURM was configured to run a prolog script after nodes have been allocated, but before the user's job script is run. This prolog script records the counts from the Lustre procfile (*/proc/fs/lustre/llite/lustre-file-system/stats*) at that time, for each node in the allocation. After the job script completes, slurm runs an epilog script. For each node in the allocation, the epilog script extracts the counters from the procfile, and calculates the difference between the "after" and "before" values for each counter. These per-node totals are then summed to obtain the total for the job. This total is stored in an RDMS database, which we queried for this data collection.

The *Aggregate Job Statistics* were collected on `Cab` for three years (April 2015 – March 2018), whereas on `Quartz`, they were collected for one year (April 2017 – March 2018).

B. Time-Series Job Statistics

Time series data on Lustre file system usage was gathered on the Lustre server nodes via the Lustre JobStats feature [20], [38], [32]. JobStats includes a job ID in every request the Lustre client sends to the servers. The Lustre server records statistics describing the requests received per Job ID [30], [33]. As a result, file system requests which are handled entirely by VFS, or which are satisfied by cached data on the client node, are not reflected in JobStats data. These statistics are gathered per-server, so the total I/O for a job is the sum of the values reported by all servers. File system `read()` and `write()` related requests include a count of bytes to be transferred. JobStats records the number of such requests received, the minimum and maximum byte count seen in requests received so far, and the sum of bytes transferred. If no requests for a given job ID are received for 600 seconds, statistics for that job ID are discarded. The absence of a job ID in the statistics on a server means the server received no requests for that job within the last 600 seconds, and so is equivalent to 0 valued counters.

We collected data from the servers using Telegraf [10] and a customized lustre2 plugin which samples the statistics by reading */proc* files Lustre exports. The proc files were */proc/fs/lustre/mdt/*/job_stats* on Lustre metadata servers, and */proc/fs/lustre/obdfilter/*/job_stats* on Lustre object storage servers. We took one sample every 60 seconds. The data gathered by Telegraf was stored in influxdb [9]. The raw samples were dumped from influxdb as CSV for analysis.

The specific statistics used are:

- ***MDS*** - *jobstats_create, jobstats_mkdir, jobstats_mknod, jobstats_open, jobstats_rename, jobstats_rmdir, jobstats_unlink*
- ***OSS*** - *jobstats_read_bytes, jobstats_read_calls, jobstats_write_bytes, jobstats_write_calls*
- *jobid, time* - Every statistic has a job ID and timestamp attached to it.

Time-Series Job Statistics were collected on `Quartz` for a period of 34 days (June 7, 2018 – July 10, 2018).

IV. Analysis

Jobs running for less than 24 hours were considered in our study. At LLNL, for longer-running jobs, Dedicated Access Time (DAT) is given. From our dataset, we saw that the percentage of cumulative compute time by DAT jobs was 2.23% and 8.74% on *Cab* and *Quartz* respectively. Therefore, the studied non-DAT jobs occupied more than 91% of the clusters over a period of years, which is a significant percentage of the total resource.

On *Cab*, the aggregate job statistics were collected for a period of three years (April 2015 to March 2018). *2,854,478* total jobs ran during this period. The number of unique users which ran the jobs is *994*.

On *Quartz*, the aggregate job statistics were collected for a year (April 2017 to March 2018). *1,401,897* jobs ran during the one year and there were *584* unique users.

A. Duration of Jobs & Number of Nodes Used

To gain insight into the behavior of jobs in terms of their duration and the number of nodes used, we plot Cumulative Distribution Function (CDF) for both of these metrics for Cab and Quartz in Figure 2.

As seen from Figures 2a and 2b, more than 90% of jobs run for less than 2 hours on both Cab and Quartz. For number of nodes used, Figures 2c and 2d show that 90% of the jobs use less than 100 nodes.

Table II shows the detailed statistics for duration and number of nodes allocated.

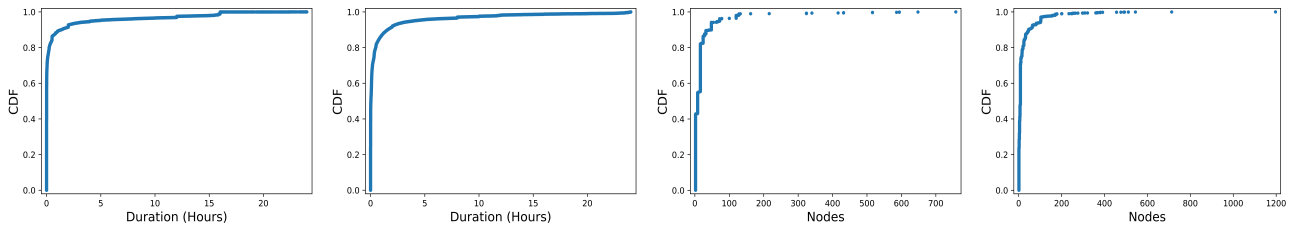
Metric	Cluster	min	max	mean	median
Duration (hours)	Cab	0.0003	23.9	0.8	0.003
	Quartz	0.0003	23.9	0.85	0.03
#Nodes	Cab	2	758	19.7	8
	Quartz	1	1197	19.8	8

TABLE II: Statistics for duration and nodes for all jobs.

Observation*: A huge effort has already been given in HPC storage systems to optimize I/O performance for long running jobs [26], [29]. However, we see that the majority of jobs on a representative real-world system consist of short-running jobs which do not occupy a lot of nodes on the system. Therefore, there should be an equal effort to optimize the I/O performance of small jobs.*

B. Distribution of Read-intensive and Write-intensive Jobs

We group jobs by users and analyze the read and write percentage of jobs run by them. This is shown in Figure 3.



(a) Cab: CDF Duration **(b)** Quartz: CDF Duration **(c)** Cab: CDF #Nodes **(d)** Quartz: CDF #Nodes

Fig. 2: Cumulative Distribution Function for Duration and #Nodes in Cab and Quartz.

We find that some users run purely write-intensive workloads and some users run only read-intensive workloads.

***Observation**: We observe that read-intensive and write-intensive jobs are distributed evenly across users. Previously, a lot of work has been done to optimize performance for write-intensive workloads. However, with the increase in machine learning workloads, which are predominantly read-intensive, there has been an overall rise in number of read-intensive workloads. Therefore, there should be an equal focus in optimizing both reads and writes in a parallel file system.*

C. Efficient and Inefficient Writes

Jobs with inefficient writes are jobs which write a large amount of data but whose number of bytes per write call is very low. We understand that small amount of writes per write call can be aggregated and Lustre does optimize these accesses before sending them to the storage servers. However, we observed that I/O performance is reduced for those write-intensive applications which perform low writes per call. We calculate the mean of bytes written by all jobs performing I/O. We also calculate the mean value of bytes written per call for all the jobs performing I/O.

Jobs with inefficient writes have total bytes written greater than the mean total write bytes across all jobs and bytes written per call is less than the mean value of writes per call across all jobs. Jobs with efficient writes have both write bytes as well as the bytes written per write call greater than the respective mean values across all jobs.

- *Jobs with inefficient writes:* (bytes written $>$ mean bytes written) and (bytes written per call $<$ mean bytes written per call)
- *Jobs with efficient writes:* (bytes written $>$ mean bytes written) and (bytes written per call $>$ mean bytes written per call)

We first see how many jobs had inefficient writes and then we focus on the users by grouping jobs by user IDs. The statistics for write bytes and write bytes per call on both Cab and Quartz are shown in Table III.

- *Classification by Jobs:* On Cab, out of 2,563,299 jobs which write more bytes than the mean value, 1,654,938 jobs (64.6%) had inefficient writes. On Quartz, out of 1,295,473 jobs, the number of jobs with inefficient writes was 893,462 (69%). The number of jobs with efficient writes on Cab and Quartz were 869,046 and 277,911 respectively.
- *Classification by Users:* On Cab, out of the 294 users whose jobs write more than the mean value of write bytes, the number of users performing inefficient writes was 138 (46.9%) and the number of users performing efficient writes was 62 (21%). On Quartz, the number of users performing inefficient and efficient writes were 111 (66%) and 57 (34%) users respectively out of 168 users who write more than the mean value of write bytes.

***Observation**: The number of jobs and correspondingly the number of users who perform efficient writes is very less. Therefore, HPC application developers should be trained to write optimal number of bytes per write call so that the applications can achieve better I/O performance.*

D. Relationship between Metadata and I/O

We sum all metadata operations in a job (*open, close, mknod, mkdir, link, unlink, rmdir, and rename*) and compare the sum with the write bytes for that job. The log-scale values are plotted and are shown in Figure 4. In both Cab and Quartz, it is seen that the number of metadata operations become larger for larger number of bytes written.

The best correlation coefficient, R, can be found for *mkdir* and *mknod* ($R = 0.93$) indicating a strong correlation between file creations and write operations.

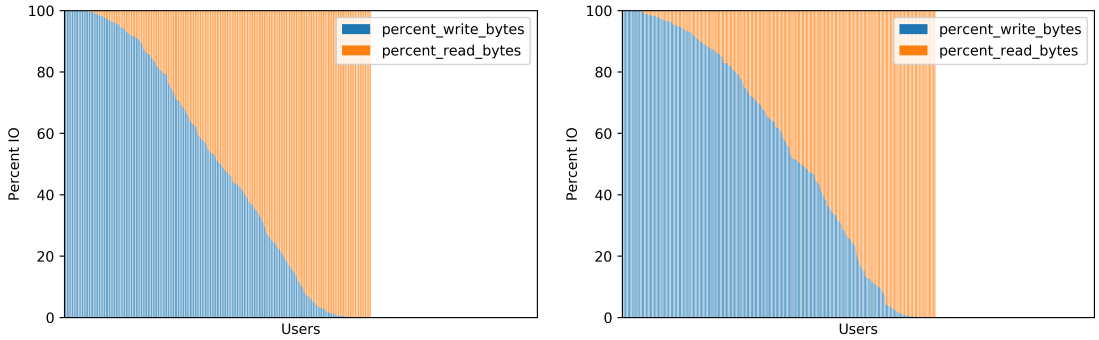
***Observation**: There is a positive correlation between metadata operations and writes, specifically for mkdir and mknod. Therefore, to improve the I/O performance, metadata operations need to be handled carefully by the metadata servers.*

E. Behavior of Metadata Servers

To manage the increase in metadata, Lustre incorporates distributed namespace (DNE) [19] - more than 1 MDTs in large HPC storage systems. Here, Lustre has 14 MDTs. The number of file opens and close requests which are being handled by different MDTs are shown in Figure 5.

It is seen in Figure 5 that the number of file opens that being handled is consistent with the number of jobs and is expected. However, we observe that not all files which are opened are closed. Moreover, the number of file open requests handled by different MDTs are different.

***Observation**: Not all files which are opened are closed which can lead to sub-optimal metadata performance due to*



(a) Cab: Percentage I/O by user.

(b) Quartz: Percentage I/O by user.

Fig. 3: Read and write percentage of users on Cab and Quartz.

Classification	Metric	Cluster	min	max	mean	median
Jobs	Write bytes	Cab	81 KB	474 TB	14.4 GB	39 GB
		Quartz	40 Bytes	597 TB	18.1 GB	146 GB
	Write bytes per call	Cab	1 Byte	820 MB	127 KB	390.9 KB
		Quartz	1.0 Byte	1.6 GB	219 KB	11.5 MB
Users	Write bytes	Cab	906 KB	4.8 PB	41.4 TB	7.4 TB
		Quartz	424 KB	4 PB	43.4 TB	27.1 TB
	Write bytes per call	Cab	178 KB	30.5 GB	184.7 MB	123.9 MB
		Quartz	899 KB	108.9 GB	350.4 MB	86.2 MB

TABLE III: Statistics for write bytes and write bytes per call for all jobs and users performing I/O on Cab and Quartz.

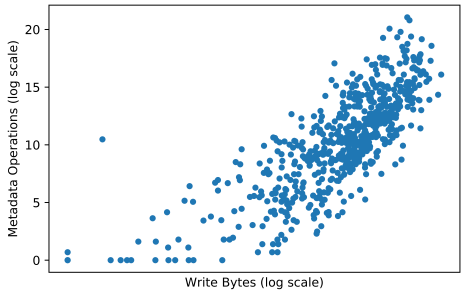


Fig. 4: #Metadata operations vs #Write operations.

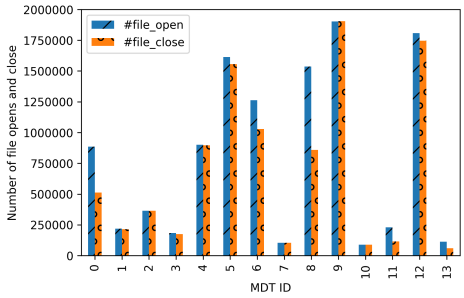


Fig. 5: #File opens and close handled by different MDTs .

dangling file pointers. Therefore, HPC users should be trained to always close open files. Additionally, there is a scope of working on balancing the metadata load across metadata targets for improving the overall I/O performance.

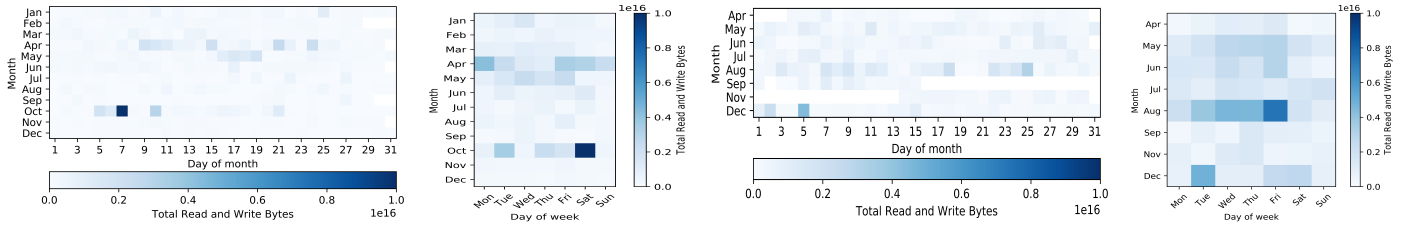
F. Temporal Analysis of I/O Traffic

For this analysis, we asked three questions of our dataset.

a) Is there any trend in I/O activity for particular months, days of the month, or days of the week?:
Are more I/O intensive jobs run during the weekend? Do people run less number of jobs on holidays? To answer these questions, we plotted heat maps showing the I/O traffic over the whole data collection period. Due to space constraints, we only show the year 2017 in Figure 6. As seen in the heat maps, there was no particular trend shown by I/O traffic. Therefore, I/O traffic from jobs cannot be predicted by the job start time with respect to the calendar or holidays.

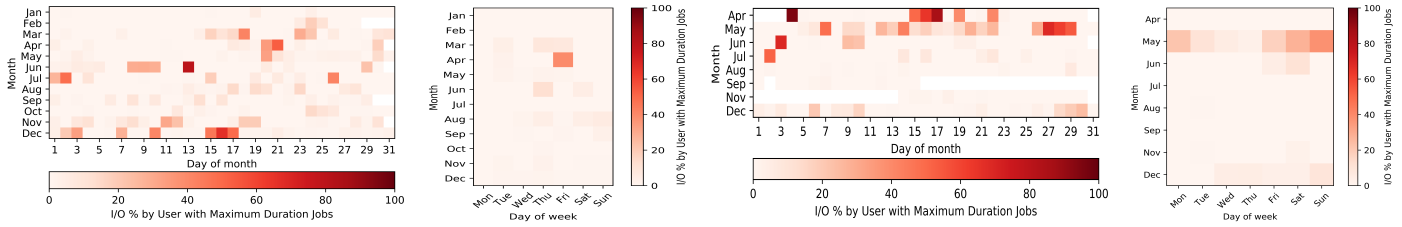
b) How much does a job which runs for the maximum duration affect the overall I/O traffic in a month, day, or a day of the week?:
We explored whether a job which ran for a long period tended to be responsible for a large portion of the I/O traffic in a system. We chose the jobs which ran for the longest period of time in a day, or a day of the week. We then calculated the percentage of total system I/O traffic contributed by each of those jobs. Figure 7 shows the contribution of a job which runs for the maximum duration to the overall traffic in a particular day and day of the week in the year 2017. There are few days where the contribution is significant, but overall, no significant effect was seen.

c) How much does a user with the highest bytes read or written in a day or a day of the week affect the overall I/O traffic?:
Our last question was how much does one user's I/O contribute to the overall I/O traffic in the system. To help answer this question, we found the users who performed the maximum I/O (highest bytes read or written) on a particular day or a day of the week. Then, we calculated the percentage of total



(a) Cab: I/O Per Day **(b)** Cab: I/O Per Day of Week **(c)** Quartz: I/O Per Day **(d)** Quartz: I/O Per Day of Week

Fig. 6: I/O Heat Map for 2017 per day and per day of the week in Cab and Quartz.



(a) Cab: Per Day **(b)** Cab: Per Day of Week **(c)** Quartz: Per Day **(d)** Quartz: Per Day of Week

Fig. 7: Contribution of Job with Maximum Duration on I/O for a day and a day of the week in 2017.

system I/O these users contributed to that day or day of the week. Figure 8 shows the contribution of users with maximum I/O on the overall I/O traffic for the year 2017. As can be seen, these users have a very significant impact on the overall I/O of the system. Therefore, job schedulers ideally would schedule other jobs with little or no I/O while these users run their jobs to reduce I/O contention and job run time.

***Observation**: There is no particular trend of I/O corresponding to a month, day of a month, or day of a week. Therefore, HPC I/O intensive jobs cannot be less I/O contended if submitted during a particular time. As expected, the I/O during a particular time when multiple I/O intensive jobs run on the system, is dominated by the job which does the maximum I/O rather than the job which runs for the maximum duration. Therefore, I/O optimizations should not be focused for long running jobs, and there is no correlation between the duration of a job and the amount of I/O requests that the job generates.*

G. Temporal distribution of I/O within jobs' runs

More than 53% of the jobs submitted to the clusters perform some I/O. But how are the I/O operations distributed across the jobs' run time?

All 16,795 jobs in the time-series dataset performed some I/O (the collection method excludes jobs without I/O). Of those jobs, 11,425 performed either read, write, or both operations. 10,443 jobs performed write operations. Since the time-series dataset records reflect I/O performed during each one-minute span, we can determine what percentage of the minutes during a job's run saw some I/O performed. We will call this percentage "I/O share". Note that during a given minute a single byte written or read causes that minute to be included in the I/O share, even though the I/O may have taken only a small fraction of a second. Figure 9 shows the I/O share for jobs which performed read, write, or both operations. The mean I/O share for these jobs is 78.8%. I/O share for jobs performing write operations is shown in Figure 10. The mean is 80.4%. Both graphs in Figures 9 and 10 are similar, therefore write operations dominate the I/O performed by jobs.

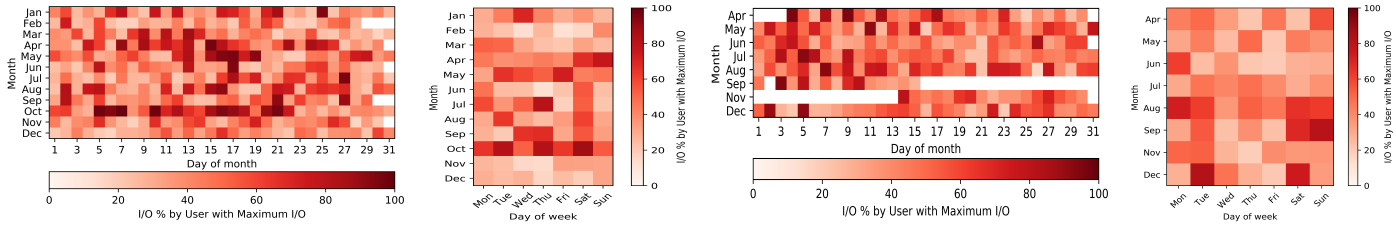
***Observation**: On average, jobs which perform I/O spread I/O activities across 78.8% of their runtime. Therefore, I/O optimizations cannot be focused at only certain time instances of job runtime. We need to work on developing I/O optimizations that aim to lower I/O contention and improve the overall I/O performance for most of the duration of application runtime.*

H. Relationship between write bytes and memory

Figure 11 shows the write pattern for 3 randomly chosen jobs. It is seen that the write bytes over time is periodic. This observation holds true for most I/O intensive jobs in a HPC storage system. Therefore, we assume that the bursts represent writing a single file and we add up the write bytes to get the size of the file. This single file could represent a checkpoint of the application data.

In the Quartz time-series dataset, 16,795 jobs were recorded. The total memory in one node in Quartz is 128 GB. We calculate the size of memory as 128 GB $*$ job node count. Only 170 jobs wrote in bursts larger than 50% of memory. 16,485 jobs wrote bursts which are smaller than than 15% of memory, while 16,173 jobs wrote bursts smaller than 5% of memory.

***Observation**: More than 95% of applications write in bursts of size less than 5% memory. Therefore, memory size is not a good predictor of write burst size as is used in many previous I/O optimization works [22]. We need better prediction approaches, such as [11] to predict I/O bursts.*



(a) Cab: Per Day **(b)** Cab: Per Day of Week **(c)** Quartz: Per Day **(d)** Quartz: Per Day of Week

Fig. 8: Contribution of User doing maximum I/O (highest read/write bytes) with respect to total I/O on the system in 2017 on a day and a day of the week's I/O.

Fig. 9: Percent minutes 11,425 jobs performed any I/O.

Fig. 10: Percent minutes 10,443 jobs performed at least one write.

Next, we inspect the write burst patterns for all I/O jobs in Quartz. First, burst size is compared to memory size. For every job, the I/O duration of the job is divided into 5 categories.

- `percent_LessThan1`: Percent of the total I/O duration when the job writes bursts are $<$ 1% memory.
- `percent_1To5`: %Duration of the total I/O duration when the job writes bursts are $\geq$ 1% and $<$ 5% memory.
- `percent_5To10`: %Duration of the total I/O duration when the job writes bursts are $\geq$ 5% and $<$ 10% memory.
- `percent_10To50`: %Duration of the total I/O duration when the job writes bursts are $\geq$ 10% and $<$ 50% memory.
- `percent_50To100`: %Duration of the total I/O duration when the job writes bursts are $\geq$ 50% and $\leq$ 100% memory.

Figure 12 shows how much of the job time is associated with different sizes of write bursts, possibly to checkpoint data. It is clear from the figure that most of the jobs spend 100% of the I/O minutes in utilizing less than 1% memory, while there are few jobs which spend all I/O minutes writing bursts in sizes between 10 – 50% of memory.

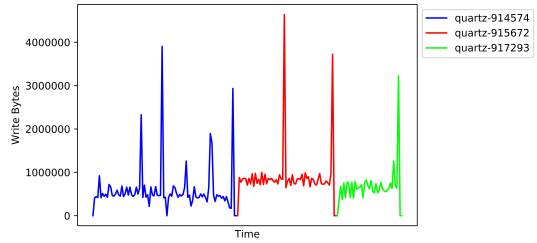


Fig. 11: Write Bytes written over time by 3 random jobs in Quartz.

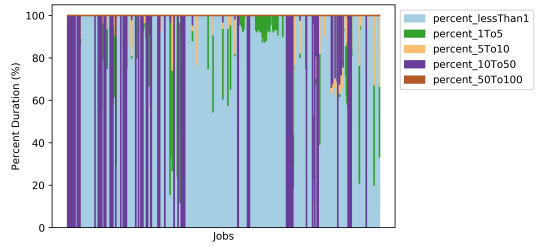


Fig. 12: % I/O duration vs. write burst size as % of memory.

Observation: *90% of jobs never write bursts larger than 1% of memory size. Therefore, we can ideally use more portions in the memory of an HPC storage system to increase the prefetching capacity and improve the overall I/O performance.*

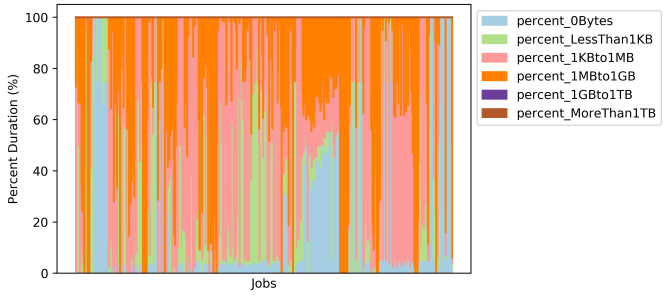


Fig. 13: Percent I/O duration vs size of write bursts.

We also look at the absolute size of the write bursts and what

percentage of the job I/O duration is associated with different sizes of bursts. Similar to Figure 12, Figure 13 shows the percentage of I/O time during which jobs issue different sizes of bursts. This is for all jobs which performed I/O.

The categories of data sizes are:

- `0 Bytes`
- `Less than 1 KB`
- `Between 1 KB and 1 MB`
- `Between 1 MB and 1 GB`
- `Between 1 GB and 1 TB`
- `More than 1 TB`

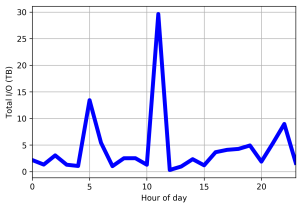
Our analysis shows that 73% jobs spend greater than 50% of their I/O time to write data bursts whose size is between 1 KB and 1 MB. There are many jobs which spend more than 90% time writing bursts of size 1 MB to 1 GB.

***Observation**: Most jobs write burst data in the range of few kilobytes for the majority of their I/O duration. In the past, I/O optimizations have worked on large checkpoint data. However, we observe that there is a significant portion of checkpoint data with small writes. Therefore, I/O optimizations should also be done for burst write requests having small number of bytes.*

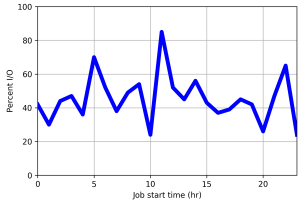
I. Demystifying I/O Contention

Inspired from [29], Figure 14a shows the total amount of data which is transferred at different hours of the day. We observe that the largest amount of I/O activity is performed by runs which start at 5AM and 11AM local time.

In Figure 14b, we plot the percentage of I/O time for applications across different hours of the day. The percentage of I/O time of an application is plotted as percentage of the maximum I/O time among all runs which perform similar I/O behavior to normalize it across applications. However, we observe that runs started at 5AM and 11AM have the highest percentage of I/O time due to the high I/O activity during this time.



(a) Total I/O for every hour throughout the day.



(b) Percentage of I/O time normalized for similar jobs starting at different hours throughout the day.

Fig. 14: Plotting I/O behavior of application for every hour throughout the day.

***Observation**: I/O contention adversely affects the I/O performance for jobs, resulting in similar applications spending more time doing I/O during the time period when other applications are also performing considerable I/O. This makes I/O performance the bottleneck for improving the overall performance of an HPC application. Therefore, significant effort is required to handle I/O contentions in the HPC centers.*

V. Discussion

The analysis of the server level statistics gave interesting insights into HPC application I/O behavior. The study though focused on Livermore Computing resources, the insights can be extended for other HPC centers. The study is conducted on a real-world deployment of Lustre file system, which is one of the most popular parallel file systems used in the top 100 supercomputers [1]. The jobs which are studied in this paper are representative of other high performance computing centers, like National Energy Research Scientific Computing Center (NERSC) [29], and Oak Ridge Leadership Computing Facility (OLCF) [26].

A. Lessons for HPC admininistrators

- There is an equal distribution of reads and writes per user in the HPC storage system. Therefore, equal focus needs to be given on optimizing both reads and writes in a parallel file system.
- The mean duration of jobs is less than 52 minutes which suggests that job scheduling and I/O contention strategies should be developed for shorter duration jobs rather than jobs which run for many hours.
- It also seems to contradict the conventional wisdom that defensive I/O is the primary use of HPC file systems, which may explain the very small number of jobs which wrote bursts larger than 1% of memory.
- A small number of jobs generated most of the load on the file system. Focusing on improving the I/O behavior of jobs which perform maximum I/O will be more beneficial for overall I/O performance than focusing on jobs which run for long durations.
- Very few applications perform efficient writes to the file system. There are applications which write burst sizes greater than 50% of memory; these may be checkpoints.
- Effort should be made in identifying periods of I/O contention in different HPC centers and users should be educated to submit I/O intensive jobs outside of those times to improve I/O performance of the overall system. Moreover, job schedulers can be made more intelligent to detect I/O contention periods.
- There should be optimizations for balancing the metadata load across the metadata servers which can adversely impact the I/O performance.

B. Lessons for HPC users

- Users who run write-intensive workloads on parallel file system need to perform efficient writes. Only 22% users perform efficient writes, which degrades the I/O performance of the entire system. The users can get better I/O performance by performing more write bytes per write function call.
- Starting an I/O intensive job at particular time instants can be beneficial in facing less I/O contention which would improve the overall performance of the application. Therefore, HPC users should know which periods of the day are bad for submitting I/O intensive applications.

- A lot of memory remains unused for I/O operations. Therefore, HPC application developers can improve I/O performance by prefetching data into memory.

VI. Related Work

We describe below the chronological order of work which focus on studying I/O behavior of workloads.

One of the earliest work in analyzing I/O characteristics was done by Pasquale et al. [27] where they studied the production workload of San Diego Supercomputing Center's Cray YMP. I/O analysis of the I/O intensive applications revealed that I/O patterns are predictive. Nieuwejaar et al. [23] also studied file access characteristics on parallel file system in 1996. Hus et al. [8] in 2001 focused on analyzing I/O behavior from the application' perspective, whereas our work emphasizes on the system side I/O behavior in an application-agnostic manner. Hsu et al. in 2003 [7] studied the I/O traffic in personal computers and server workloads. Even on small-scale personal computers, the analysis of I/O traffic showed similar bursty patterns as our result on large-scale LLNL supercomputers.

In 2004, Wang et al. [39] analyzed workloads in a LLNL cluster. However, they focused on studying application traces to understand the types and sizes of file requests sent by application. Thereafter in 2009, Carns et al. did three studies [4], [14], [3], all targeting applications in very large scale storage systems at Argonne National Laboratory (ANL). In these three works, they studied application traces from Darshan I/O characterization tool [13], how components work together to provide I/O services to applications. 2009 was one of the earliest times when petascale I/O workloads were studied, but all of these studies targetted application traces. The first work to study failure logs in a large scale storage system was also done in 2009 by Taerat et al. [37], where they analyzed Blue Gene/L failure log data at both system and application-levels.

In 2010, Zhao et al. [43] and Shipman et al. [36] studied the Lustre file system. While, Zhao et al. [43] worked on a prediction model to accurately predict I/O bandwidth in Lustre file system, Shipman et al. [36] talked about how the world's largest Lustre file system in 2010 was deployed in the Spider system at Oak Ridge National Laboratory (ORNL). Oral et al. [25] extended Shipman et al.'s work [36] at ORNL by discussing the lessons learned from deploying large-scale parallel file systems in Oak Ridge Leadership Computing Facility (OLCF).

Kim et al. in 2010 [12] studied I/O needs for over 4 million applications on HPC clusters. Carns et al. [2] in 2011, studied an application-biased I/O characterization by performing a two month study on Interpid. In 2012, Saini et al. [34] studied the I/O behavior of five NASA applications on Lustre file system. In 2016, Luu et al. [21] and Gunasekaran, et al. [5] discussed the application level I/O behavior at production scale at ANL and ORNL respectively. Using application-level behavior, Liu et al. [16] in 2016, and Wyatt et al. [41] in 2017 developed machine learning algorithms to coordinate I/O traffic on large scale shared storage systems. All of these works focus on managing I/O per-application basis without considering system-side statistics.

Lim et al. [15] in 2017 studied system-side statistics, but only considering metadata operations without being application-agnostic. We have taken this work one step further by studying both metadata as well as storage server statistics without being dependent on application characteristics. Lockwood et al. [18] built TOKIO in 2018 which uses analysis results to quantify the degree of I/O contention and the benefit to users to migrate to burst buffers. The same authors in 2018 provided an extensive analysis [17] from the tools used in TOKIO. IOMiner [40] was developed by Wang et al. which provided a unified interface to query I/O statistics and analyze them. Zhou et al. [44] in 2018 and Yang et al. [42] in 2019 use the application-level I/O characteristics to build an in-memory computing framework and an end-to-end I/O monitoring solution. In 2020, Patel et al. [29], [28], Paul et al. [31] and Kim et al. [11] work towards demystifying file access patterns using file level statistics.

VII. Conclusion

Improving I/O performance has become the most important factor in modern I/O bound HPC applications. Therefore, understanding the I/O behavior of HPC applications is very important for system administrators, file system developers, and HPC users. This paper collected Lustre file system server level statistics from two clusters, Cab and Quartz, at the Lawrence Livermore National Laboratory, for a period of three years and analyzed the statistics in an application-agnostic manner. Our studies have indicated interesting results which show that due to the increase in popularity of machine learning jobs, the HPC jobs now have an even distribution of write-intensive and read-intensive jobs, showing the importance of giving equal priority in improving file system read and write performance. Our analysis also led us to believe that there should be focus on I/O optimizations for jobs which run for short duration. Also, there should be efforts to educate HPC users to develop applications which perform efficient writes. Moreover, the metadata spread across metadata servers are unbalanced and there should be efforts to balance the load or migrate metadata operations to less loaded metadata server. Much effort is also needed to mitigate the effect of I/O contention that adversely impacts the I/O performance of the entire system. We believe that our analysis will help all HPC practitioners to build better file systems and utilize it more effectively.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CONF-812199. This work is sponsored in part by the National Science Foundation under grants CCF-1919113, CNS-1405697, and OAC-2004751.

REFERENCES

[1] T. 500. Top 500 list - june 2020. Accessed: July 16 2020.

[2] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. Understanding and improving computational science storage access through continuous characterization. *ACM TOS*, 7(3):8, 2011.

[3] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig. Small-file access in parallel file systems. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–11. IEEE, 2009.

[4] 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*, pages 1–10. IEEE, 2009.

[5] R. Gunasekaran, S. Oral, J. Hill, R. Miller, F. Wang, and D. Leverman. Comparative i/o workload characterization of two leadership class storage clusters. In *Proceedings of the 10th Parallel Data Storage Workshop*, pages 31–36. ACM, 2015.

[6] J. L. Hennessy and D. A. Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[7] W. W. Hsu and A. J. Smith. Characteristics of i/o traffic in personal computer and server workloads. *IBM Systems Journal*, 42(2):347–372, 2003.

[8] W. W. Hsu, A. J. Smith, and H. C. Young. I/o reference behavior of production database workloads and the tpc benchmarks—an analysis at the logical level. *ACM TODS*, 26(1):96–143, 2001.

[9] InfluxData. Influxdb. Accessed: April 1 2019.

[10] InfluxData. Telegraf. Accessed: April 1 2019.

[11] S. Kim, A. Sim, K. Wu, S. Byna, Y. Son, and H. Eom. Towards hpc i/o performance prediction through large-scale log analysis. In *Proceedings of the 29th HPDC*, pages 77–88, 2020.

[12] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Z. Zhang, and B. W. Settlemyer. Workload characterization of a leadership class storage cluster. In *2010 PDSW*, pages 1–5. IEEE, 2010.

[13] A. N. Laboratory. Darshan - hpc i/o characterization tool. Accessed: May 12 2019.

[14] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock. I/o performance challenges at leadership scale. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12. IEEE, 2009.

[15] S.-H. Lim, H. Sim, R. Gunasekaran, and S. S. Vazhkudai. Scientific user behavior and data-sharing trends in a petascale file system. In *SC*, page 46. ACM, 2017.

[16] Y. Liu, R. Gunasekaran, X. Ma, and S. S. Vazhkudai. Server-side log data analytics for i/o workload characterization and coordination on large shared storage systems. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 819–829. IEEE, 2016.

[17] G. K. Lockwood, S. Snyder, T. Wang, S. Byna, P. Carns, and N. J. Wright. A year in the life of a parallel file system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, page 74. IEEE Press, 2018.

[18] G. K. Lockwood, N. J. Wright, S. Snyder, P. Carns, G. Brown, and K. Harms. Tokio on clusterstor: connecting standard tools to enable holistic i/o performance analysis. In *2018 Cray User Group*, 2018.

[19] Lustre. Lustre - distributed name space. Accessed: July 15 2020.

[20] Lustre. Lustre jobstats. Accessed: April 1 2019.

[21] 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 HPDC*, pages 33–44. ACM, 2015.

[22] R. McKenna, S. Herbein, A. Moody, T. Gamblin, and M. Taufer. Machine learning predictions of runtime and io traffic on high-end clusters. In *2016 CLUSTER*, pages 255–258. IEEE, 2016.

[23] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. L. Best. File-access characteristics of parallel scientific workloads. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1075–1089, 1996.

[24] OpenSFS and EOFS. Lustre file system. Accessed: March 11 2019.

[25] S. Oral, J. Simmons, J. Hill, D. Leverman, F. Wang, M. Ezell, R. Miller, D. Fuller, R. Gunasekaran, Y. Kim, et al. Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems. In *SC'14*, pages 217–228. IEEE, 2014.

[26] S. Oral, S. S. Vazhkudai, F. Wang, C. Zimmer, C. Brumgard, J. Hanley, G. Markomanolis, R. Miller, D. Leverman, S. Atchley, et al. End-to-end i/o portfolio for the summit supercomputing ecosystem. In *Proceedings of SC*, pages 1–14, 2019.

[27] B. K. Pasquale and G. C. Polyzos. A static analysis of i/o characteristics of scientific applications in a production workload. In *Supercomputing'93: Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, pages 388–397. IEEE, 1993.

[28] T. Patel, S. Byna, G. K. Lockwood, and D. Tiwari. Revisiting i/o behavior in large-scale storage systems: the expected and the unexpected. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2019.

[29] T. Patel, S. Byna, G. K. Lockwood, N. J. Wright, P. Carns, R. Ross, and D. Tiwari. Uncovering access, reuse, and sharing characteristics of i/o-intensive files on large-scale production {HPC} systems. In *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*, pages 91–101, 2020.

[30] A. K. Paul, R. Chard, K. Chard, S. Tuecke, A. R. Butt, and I. Foster. Fsmonitor: Scalable file system monitoring for arbitrary storage systems. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11. IEEE, 2019.

[31] A. K. Paul, O. Faaland, A. Moody, E. Gonsiorowski, K. Mohror, and A. R. Butt. Understanding hpc application i/o behavior using system level statistics, 2019.

[32] A. K. Paul, A. Goyal, F. Wang, S. Oral, A. R. Butt, M. J. Brim, and S. B. Srinivasa. I/o load balancing for big data hpc applications. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 233–242. IEEE, 2017.

[33] A. K. Paul, B. Wang, N. Rutman, C. Spitz, and A. R. Butt. Efficient metadata indexing for hpc storage systems. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 162–171. IEEE, 2020.

[34] S. Saini, J. Rappleye, J. Chang, D. Barker, P. Mehrotra, and R. Biswas. I/o performance characterization of lustre and nasa applications on pleiades. In *2012 19th International Conference on High Performance Computing*, pages 1–10. IEEE, 2012.

[35] SchedMD. Slurm workload manager. Accessed: April 1 2019.

[36] G. Shipman, D. Dillow, S. Oral, F. Wang, D. Fuller, J. Hill, and Z. Zhang. Lessons learned in deploying the world's largest scale lustre file system. In *The 52nd Cray user group conference*, 2010.

[37] N. Taerat, N. Naksinehaboon, C. Chandler, J. Elliott, C. Leangsuksun, G. Ostrouchov, S. L. Scott, and C. Engelmann. Blue gene/l log analysis and time to interrupt estimation. In *2009 International Conference on Availability, Reliability and Security*, pages 173–180. IEEE, 2009.

[38] B. Wadhwa, A. K. Paul, S. Neuwirth, F. Wang, S. Oral, A. R. Butt, J. Bernard, and K. W. Cameron. iez: Resource contention aware load balancing for large-scale parallel file systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 610–620. IEEE, 2019.

[39] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. Miller, D. Long, and T. McLarty. File system workload analysis for large scale scientific computing applications. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2004.

[40] T. Wang, S. Snyder, G. Lockwood, P. Carns, N. Wright, and S. Byna. Iominer: Large-scale analytics framework for gaining knowledge from i/o logs. In *2018 CLUSTER*, pages 466–476. IEEE, 2018.

[41] M. Wyatt, S. Herbein, T. Gamblin, A. Moody, A. Dong, and M. Taufer. From job scripts to resource predictions: Paving the path to next-generation hpc schedulers. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2017.

[42] B. Yang, X. Ji, X. Ma, X. Wang, T. Zhang, X. Zhu, N. El-Sayed, H. Lan, Y. Yang, J. Zhai, et al. End-to-end i/o monitoring on a leading supercomputer. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 379–394, 2019.

[43] T. Zhao, V. March, S. Dong, and S. See. Evaluation of a performance model of lustre file system. In *2010 Fifth Annual ChinaGrid Conference*, pages 191–196. IEEE, 2010.

[44] P. Zhou, Z. Ruan, Z. Fang, M. Shand, D. Roazen, and J. Cong. Doppio: I/o-aware performance analysis, modeling and optimization for in-memory computing framework. In *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 22–32. IEEE, 2018.