Understanding HPC Application I/O Behavior Using System Level Statistics

Arnab K. Paul
Virginia Tech

Olaf Faaland
Lawrence Livermore National Laboratory

Adam Moody
Lawrence Livermore National Laboratory

Elsa Gonsiorowski
Lawrence Livermore National Laboratory

Kathryn Mohror
Lawrence Livermore National Laboratory

Ali R. Butt
Virginia Tech

1 INTRODUCTION

The current trend for high performance computing (HPC) systems is that 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 [3]. The Lustre file system [5] is one of the most widely-used parallel file systems, supporting seven of the top ten supercomputers in the latest Top-500 list (November, 2018) [1].

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. 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.

In this poster, we collect and study file system statistics from two Livermore Computing systems with 15 PiB Lustre file systems at Lawrence Livermore National Laboratory (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 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.

2 BACKGROUND

2.1 Lustre Distributed File System

The architecture of the Lustre file system is shown in Figure 1 [6]. The *Management Server (MGS)* is responsible for storing the configuration information for the entire Lustre file system in the *Management Target (MGT)*. The *Metadata Server (MDS)* manages all the namespace operations for the file system and are stored in an *Metadata Target (MDT)*. *Object Storage Servers (OSSes)* provide the storage for the file contents on one or more *Object Storage Target (OST)*s. Applications access the file system data via *Lustre clients*. The internal high-speed data networking protocol for Lustre file system is is managed by the *Lustre Network (LNet)* layer.

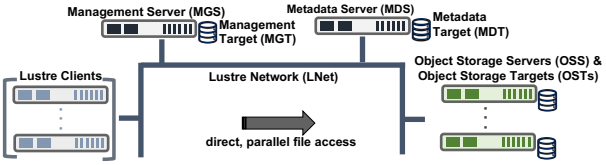


Figure 1: An overview of Lustre architecture.

2.2 Clusters

Table 1: Cluster Configurations.

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

Table 1 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.

2.3 Data Collection

2.3.1 Aggregate Job Statistics. Both `Cab` and `Quartz` use the SLURM job scheduler [7] to run a prolog and an epilog script which records the counts from the Lustre procfile (*/proc/fs/lustre/llite/lustre-file-system/stats*) for each node in the application allocation before and after a job script completes. These per-node totals are then summed to obtain the total for the job which is stored in an RDMS database and queried.

The specific statistics used are:

- *starttime, endtime, duration, uid, nodes*
- *mkdir, mknod, open, rename, rmdir, unlink*
- *read_bytes, write_bytes, read_bytes_count, write_bytes_count*
- *recv_bytes, recv_count, send_bytes, send_count*

[1]Cab was decommissioned in June 2018.

2.3.2 Time-Series Job Statistics. We collected Lustre JobStats data [4] from the servers using Telegraf [8] and a customized lustre2 plugin which samples the statistics by reading the proc files – */proc/fs/lustre/mdt/*/job_stats* on MDS, and */proc/fs/lustre/obdfilter/*/job_stats* on OSSes. We took one sample every 60 seconds. The data gathered by Telegraf was stored in influxdb [2]. The raw samples were dumped from influxdb as CSV for analysis.

The specific statistics used are:

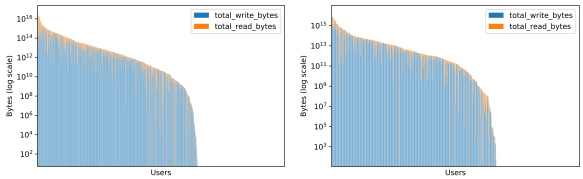
- ***MDS*** - *jobstats_create, jobstats_mkdir, jobstats_mknod, jobstats_open, jobstats_rename, jobstats_rmdir, jobstats_unlink*
- ***OSS*** - *jobstats_read_bytes, jobstats_read_calls, jobstats_write_bytes, jobstats_write_calls*
- *jobid, time* - Every statistic has a job ID and timestamp.

3 ANALYSIS

On *Cab*, the aggregate job statistics were collected for a period of three years. *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. *1,401,897* jobs ran during the one year and there were *584* unique users.

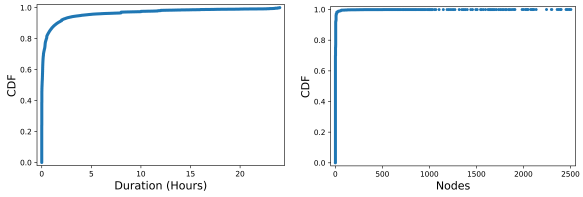
Observation 1*: Most of the users (greater then 65%) perform significant I/O in jobs, and the users mostly run write-intensive jobs.* This can be seen in Figure 2.



(a) Cab **(b) Quartz**

Figure 2: Total Reads and Writes per User (Zero values on the right side show users who performed no I/O to Lustre).

Observation 2*: 90% of jobs run for less than 2 hours and are allocated less than 100 nodes. The mean value for duration of these jobs is less than 52 minutes.* This is evident from the Cumulative Distribution Function (CDF) in Figure 3.

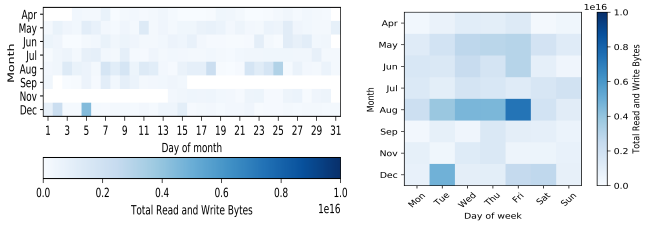


(a) Quartz: CDF Duration **(b) Quartz: CDF #Nodes**

Figure 3: Cumulative Distribution Function for Duration and #Nodes in Quartz.

Observation 3*: Less than 22% of users submitting write-intensive jobs perform efficient writes.* 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.

Observation 4*: I/O performed by a job is not correlated with month, day of the month, day of the week, or presence of a holiday.* This is shown in Figure 4.



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

Figure 4: I/O Heat Map for 2017 per day and per day of the week in Cab and Quartz.

Observation 5*: 90% of jobs never write bursts larger than 1% of memory size.* It is seen that the write bytes over time is periodic. 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. Figure 5 shows that most of the jobs spend 100% of the I/O minutes in utilizing less than 1% memory.

Figure 5: % I/O duration vs. write burst size as % of memory.

4 CONCLUSION

Our studies have indicated interesting results which show that most jobs are write-intensive, showing the importance of improving file system write performance. Our analysis also led us to believe that focus should be on jobs which run for short duration as the majority of the jobs run for less than an hour. Also, there should be efforts to educate HPC users to develop applications which perform efficient writes. This would improve I/O performance as well as help in reducing I/O contention among jobs.

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-POST-784599. This work is also sponsored in part by the NSF under the grants: CNS-1405697, CNS-1615411, and CNS-1565314/1838271.

REFERENCES

[1] T. 500. Top 500 list - novemebr 2018. https://www.top500.org/lists/2018/11/. Accessed: March 7 2019.
[2] InfluxData. Influxdb. https://github.com/influxdata/influxdb. Accessed: April 1 2019.

[3] 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.

[4] Lustre. Lustre jobstats. http://doc.lustre.org/lustre_manual.xhtml#dbdoclet.jobstats. Accessed: April 1 2019.

[5] OpenSFS and EOFS. Lustre file system. http://lustre.org/. Accessed: March 11 2019.

[6] 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 *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11. IEEE, 2019.

[7] SchedMD. Slurm workload manager. https://slurm.schedmd.com/overview.html. Accessed: April 1 2019.

[8] Telegraf. Telegraf. https://github.com/influxdata/telegraf. Accessed: April 1 2019.