Improving the Performance of Lustre File System in HPC Environments

Jaehyun Han*, Deoksang Kim* and Hyeonsang Eom*
*Department of Computer Science and Engineering
Seoul National University, Seoul, Korea
Email: {jhhan,dskim,hseom}@dcslab.snu.ac.kr

***Abstract*—As more and more data is being processed in High Performance Computing (HPC) systems scaling up in terms of the processing power and the amount of data to process, slow I/O can become a major performance bottleneck. Therefore, HPC systems usually use distributed parallel file systems which can lead to high I/O performance. However, it is still possible to enhance the performance of distributed parallel file systems by tuning the parameters or choosing the best configurations. To improve the performance of distributed parallel file systems, we first analyzed and compared the major ones being widely used in HPC environments. Then we focused on the Lustre file system for a further analysis because Lustre is one of the major ones due to its high performance. We evaluated Lustre in the direct I/O mode with various configurations and different parameter settings. Our experimental study suggests that we can obtain up to 114% performance improvements by modifying the numbers of Portal RPC threads in the OSS (Object Storage Server) and client.**

I. Introduction

Recently the computing power and the amount of data to process in HPC environments have increased tremendously, the researchers in the HPC community and government officials in some countries have started preparing for the era of exascale computing. It is expected that file I/O will be one of the major performance bottlenecks especially in exascale systems because today's file and storage systems are not being developed with sufficient consideration of the requirement of performing a huge number of I/O requests efficiently even though the amount of data to process is increasing rapidly. To meet this requirement, distributed parallel file systems which can lead to high I/O performance are usually used in HPC systems.

However, it is still possible to enhance the performance of distributed parallel file systems by tuning the parameters or choosing the best configurations. Since Lustre [1], GPFS [2], and OrangeFS [3] are typical distributed parallel file systems widely being used in HPC environments, we first analyzed and compared these distributed parallel file systems in order to eventually figure out how much performance gain could be obtained based on the difference in performance and mechanism among them.

We focused on Lustre because it is being developed quite actively as an open-source project. Before analyzing the performance of Lustre, we followed the development trends in the Lustre developer community. As a file system cluster becomes huge, the metadata server is likely to become a performance bottleneck; Lustre had supported the use of only one metadata server for a long time until recently, when the use of multiple metadata servers was proposed in the community [4]. To remove the metadata bottleneck, the developers are trying to distribute the metadata. In this paper, we present major efforts made in the community to improve the Lustre performance by removing its bottlenecks. To analyze the performance bottlenecks in Lustre, we evaluated its performance with various configurations and different parameter settings. We performed our experiments in the direct I/O mode because there have not been many studies conducted to find the bottlenecks when using direct I/O. We found some characteristics as the numbers of Lustre clients and I/O threads were changed. The throughput increases as the number of Lustre clients or I/O threads increases. For read operations, the throughput in the case of using two clients is lower than that in the case of using a single client, but with a sufficient number of clients the read throughput becomes high again. And small stripe counts (such as 1 and 2) may lead to low throughput even when the number of clients is large.

The Lustre distributed file system consists of MDSes (Meta-Data Servers), OSSes (Object Storage Servers), and Lustre clients. They use RPC (Remote Procedure Call) to communicate with one another. Every Lustre cluster has Portal RPC processes. These processes take charge of transmission of data between the server and client using the RPC protocol and recovery of erroneous transfers.

The number of Portal RPC threads in a node matches up with the number of CPU cores in that node by default. In an HPC environment, a Lustre client is actually a compute node where computation is performed usually to solve compute intensive problems. Therefore, we thought that a larger number of Portal RPC threads than necessary would disturb the computation jobs. We evaluated Lustre with the numbers of Portal RPC threads in OSSes and clients changed. Our experimental study suggests that we can obtain up to 114% performance improvements when the number of Portal RPC threads in the OSS is 8 (which is about 2/3 of the number of CPU cores) and the number of Portal RPC threads in the client is 4 (which is about 1/5 of the number of CPU cores).

II. Comparison Study and Related Work

We compared three commonly used file systems in HPC environments: Lustre, GPFS, and OrangeFS. Hedges et al.

compared the performance of Lustre with that of GPFS [5]. They claimed that GPFS outperforms Lustre by 20% or more in writing data while the reading performance is almost the same. In metadata operations, Lustre is faster in most of the cases. But in the only case of creating many files in a shared directory, GPFS is faster than Lustre.

Tanimura et al. studied shared storage systems [6]. They presented the basic I/O throughput of Lustre and OrangeFS. They found that OrangeFS was better in write throughput and Lustre was better in read throughput. Roberts compared next generation file systems for his center [7]. He claimed that OrangeFS was extremely easy to install while the setup of Lustre was complex.

Table I shows the result of the comparison in feature among the popular HPC file systems.

There have been many studies recently conducted to alleviate I/O performance bottlenecks in the Lustre file system. Previously, this file system could utilize only one MDS. But as the amount of data to process has increased, the use of a single MDS has become insufficient in performing every metadata operation. To increase the performance of the Lustre file system, Lustre developers introduced the DNE(Distributed NamespacE) technology. The DNE technology was adopted in two phases. *DNE Phase 1* had been supported since Lustre v2.4 released in 2013. In this phase, Lustre supported multiple MDSes for different directories. But the content of a directory should reside in a single server. *DNE Phase 2* was introduced as a preview on Lustre 2.6 In Phase 2, Lustre stripes the content of a directory and stores the resulting stripes into multiple MDSes. Crowe et al evaluated the DNE in various configurations [4]. They found that the metadata performance measured by `mknod` was improved by more than 200%.

Fujitsu developed FEFS (Fujitsu Exabyte File System) for their supercomputer [8]. FEFS is based on Lustre. Fujitsu made some modifications on it. When there are a huge number of clients using Lustre, processing the metadata can be a performance bottleneck. FEFS creates *local loopback* file systems for every `MPI RANK`, and then uses them as MDTs (MetaData Targets) [9]. In this approach, the file system cannot be shared among the clients that are using different `MPI RANKs`. But FEFS mitigated the performance bottleneck of processing the metadata, and thus its use led to 21,571 times performance improvements in the processing when 9,216 clients used it simultaneously. These performance improvements were measured by performing the `unlink` operations.

When a client reads a file from Lustre, the client first obtains the metadata of the file from the MDS. Using the metadata, the client makes a request for the file to a specific OSS. This mechanism causes a performance problem when a client requests a large number of small files. In HPC workloads, most files are large, and thus this is not a big problem. But since the Lustre file system has been adopted for various purposes, the performance for small file I/O has become important. To improve the small I/O performance, Lustre developers suggests DoM (Data on MDT) [10]. The DoM technique is simply to store small files on MDTs instead of OSTs (Object Storage Targets). In this way, the number of network operations can become much smaller since many accesses to OSSes can be eliminated in small file I/O. Therefore, performance bottlenecks regarding metadata can be relieved.

III. Problem & Approach

Since Lustre is a distributed parallel file system, it consists of a large number of machines. The performance of Lustre depends on its configuration and parameter setting. It is quite possible to enhance the performance of distributed parallel file systems such as Lustre by tuning the parameters or choosing the best configurations. To find the best configuration and parameter setting, we performed the evaluation with various configurations and parameter settings. Then we analyzed the evaluation result, and we found out the configurations and parameter settings that could lead to performance improvements, which eventually permits providing guidelines for performance enhancement.

Lustre accelerates processing data by performing it in parallel. To further improve its performance, we take an approach of evaluating it while varying the degree of distribution. When multiple clients use the file system simultaneously, the performance can be enhanced because it executes in a distributed manner so that it can process many requests simultaneously. It can store files striped and distributed in different OSTs. As it stripes and distributes every file to all of the OSTs, multiple requests can be made to the OSTs in a distributed fashion. This can prevent multiple requests from being made and concentrated to a small number of OSTs. Therefore in order to evaluate its performance, we changed the number of clients, that of I/O threads per client, and stripe count.

We also found every node has a Portal RPC process for RPC communication. The number of Portal RPC threads is fixed to the number of CPU cores in a specific node by default. Since the OSSes and clients need the CPUs to do jobs other than the communication, we thought the default number of Portal RPC threads could be too large. Therefore, we also evaluated the performance while changing the number of Portal RPC threads on OSSes and Clients.

IV. Evaluation

We evaluated the I/O performance of the Lustre file system with various configurations and parameter settings. We focused on the Direct I/O mode because some applications need to use this mode for file operations. But the Lustre performance under the mode has not been well studied yet.

For evaluation, we used Lustre version 2.7 with one MDS and one OSS. The OSS was configured with 4 OSTs, and each of them consisted of 10 disks (8 for capacity and 2 for parity). Since Lustre parallelizes the operations w.r.t. OSTs, this configuration was adequate to our needs. We used up to 8 clients simultaneously to run benchmarks. Each of the MDS and clients had 2 * 10-core CPUs and the OSS had 2 * 6 core CPUs. All machines were connected via a FDR (Fourteen

	Lustre	GPFS	OrangeFS (prev. PVFS2)
Block management	Object based	Shared block map	Object based
Stripe size	1MB	1MB	64KB
Metadata location	Separate	With data	With data
Metadata written by	Server	Client	Client
Cache coherency & protocol	Coherent; Distributed locking	Coherent; Distributed locking	Cache immutable
Reliability	Block RAID	Block RAID	Block RAID
License	Open Source (GPL v2)	Proprietary (IBM)	Open Source (LGPL)

TABLE I: Comparison of the popular file systems in HPC environments

Data Rate) InfiniBand switch which provides a 56Gbps link bandwidth.

A. Performance in different client configurations

To evaluate the performance in different client configurations, we changed three parameters. We changed the number of clients from 1 to 5, the number of I/O threads per client from 1 to 2, and the stripe count from 1 to 4. In this experiment, we used the `IOR` benchmark [11]. We configured the `IOR` benchmark to read and write 16GB of data sequentially. The transfer size was 64MB. We used the Direct I/O mode as mentioned above. In most of the cases, we could find that the performance is improved as the number of I/O client nodes increases or the number of threads per client increases. But in the case of the read operation, the performance dropped when two clients performed the read operation simultaneously. Yet when the number of clients increases and becomes more than two, the performance is improved in a scalable manner. In addition, the stripe count influences the performance. As the stripe count increases, indicating that the I/O parallelism becomes higher, the throughput also increases as expected. In the write experiment, when the stripe count is 1 or 2, the throughput is saturated around 800MB/s. But when the stripe count is 3 or 4, the resulting throughput is over 800MB/s. This result shows that the client scalability becomes higher as the stripe count increases.

Then we measured the performance of Lustre in two configurations. In one configuration, we used a single client while changing the number of I/O threads from 1 to 3. In the other, we changed the number of clients from 1 to 3, and used a single I/O thread for every client. In this experiment, we could find a performance drop similar to what mentioned above, which happened when two clients or threads read data simultaneously. As confirmed with the result of the experiment, the performance drop is much higher when the number of clients changes from 1 to 2 than when the number of client threads changes from 1 to 2.

B. Performance while changing the number of Portal RPC threads of OSSes and clients

In Lustre, Portal RPC is used to communicate between the server and client. The architecture of the Lustre system is illustrated in Figure 1. The number of these Portal RPC threads on each node is set to the number of cores on that machine by default. In a computer system, only one thread on each core

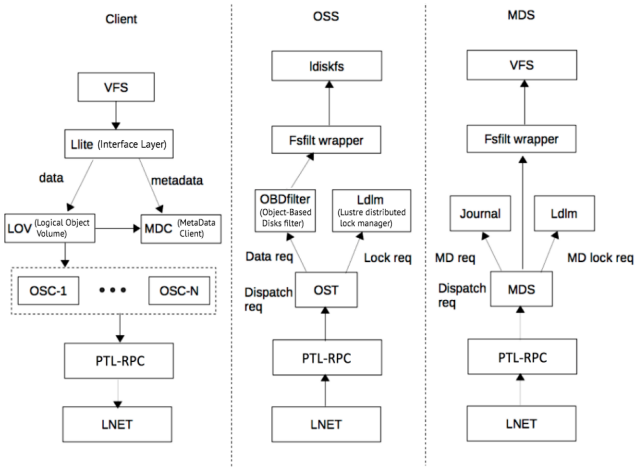

Fig. 1: Illustration of Lustre Architecture from Understanding Lustre File System Internals [12]

can execute at the same time. In an HPC system using Lustre, a client processes its own jobs and communicates with a Lustre server when data I/O is needed. An OSS also has many jobs to process, such as reading data from OSTs, and communication is just one of the jobs. Therefore we thought that having too many Portal RPC threads could be a performance bottleneck for the entire Lustre system, lowering the performance of the workloads. To find out the "best" number of Portal RPC threads for OSSes and clients, we measured the performance with changing the number of Portal RPC threads on OSSes and clients. An OSS executed on a machine with two 6-core CPUs, and by default 12 Portal RPC threads were created on the machine. A client ran on a machine with two 10-core CPUs, and by default 20 Portal RPC threads were created on the machine. We measured the performance while changing the number of OSS Portal RPC threads to 4, 8, and 12 , and the number of client Portal RPC threads to 4, 8, and 20. We used IOR to measure the performance; the transfer size was fixed to 64MB, and the stripe size was fixed to 1MB.

Figure 2 shows the throughput of Lustre using a single client with 16 I/O threads. In the legend, `s` denotes the number of Portal RPC threads in the OSS and `c` denotes the number of Portal RPC threads in the client. As shown in Figure 2, when a single client sequentially read data using multiple threads, the performance result did not show any trend. But in the case of the write operation, when the number of OSS Portal RPC threads was 8 and the number of client Portal RPC threads was

4, the performance was the best. This corresponds to a 113% improvement compared with the default configuration. We can find that the default configuration usually led to the worse performance compared with all other configurations. When the stripe count was 1, the performance difference was noticeable, but when the count was larger than 2, the performance was lower and performance difference was less distinguishable.

Figure 3 shows the throughput of Lustre using a single client with 8 I/O threads. Similar to the previous experiment, there was no performance difference for sequential reads. But for sequential writes, the best performance was achieved when the number of OSS Portal RPC threads was 8 and the number of client Portal RPC threads was 4, which is also the best configuration for the previous experiment. As a result, up to 53% performance improvements were achieved. Also since we decreased the number of I/O threads to 8, and thus overall performance was lowed compared with the previous experiment.

Figure 4 shows the throughput of Lustre using a single client with a single I/O thread. In this experiment, both the sequential read and sequential write throughputs were not influenced by the number of Portal RPC threads. Thus we may conclude that if the number of client I/O threads is lower, the performnace in that configuration is less affected by the number of Portal RPC threads.

Figure 5 shows the throughput of Lustre using 8 clients, each with 1 I/O thread. Similar to the results of the previously explained experiments, the sequential read throughput was not influenced by the number of Portal RPC threads. For sequential writes, the performance was improved by up to 71% when the number of OSS Portal RPC threads was 8 and the number of client Portal RPC threads was 4. Also in this experiment, the performance difference was most noticeable when stripe count was 1.

Figure 6 shows the throughput of Lustre using 8 clients, each with 8 I/O threads. For sequential reads, small performance differences were found as the number of Portal RPC threads changed, but the performance difference was too small, thus being negligible. For sequential writes, the throughput was lower than the default configuration except for the case where the number of OSS Portal RPC threads was 8 and the number of client Portal RPC threads was 20. Also for sequential writes, it was interesting that the throughput increased as the stripe count increased as shown at some points in the figure.

Figure 7 shows the throughput of Lustre using 8 clients, each with 16 I/O threads. For sequential writes, when the stripe count was 1, the performance difference made by changing the number of Portal RPC threads was negligible. The performance was improved by up to 20% when the number of OSS Portal RPC threads was 8 and the number of client Portal RPC threads was 4. In this experiment, the read throughput increased as the stripe count increased.

Via all of these experiments, we found that for sequential reads, the number of Portal RPC threads does not influence the read throughput. But for sequential writes, the performance difference made by changing the number is noticeable. Our experimental study suggests that we can achieve the best sequential write throughputs when using 8 Portal RPC threads per OSS (which is 2/3 of the number of system cores) and 4 Portal RPC threads per client (which is 1/5 of the number). When the stripe count was 1, the use of the configuration with a single client and 8 I/O threads led to up to 54% performance improvements; A single client with 16 I/O threads, up to 114% improvements; 8 clients, each with a single I/O thread, up to 40% improvements; 8 clients, each with 8 I/O threads, up to 12% performance decreases; and 8 clients, each with 16 I/O threads, up to 13% decreases. Different performance tendencies were observed as the stripe count or the number of clients changed. When the stripe count was 1 or 2, the resulting throughput was higher than that in the default configuration as long as the number of Portal RPC threads was low in both of the OSS and client. When the stripe count was 3 or 4, the resulting throughput was not always higher than that in the default configuration. We suspect that this was probably affected by the RAID configuration. In our OSS, the OSTs were configured as RAID 6. RAID 6 uses 2 parity blocks to store data. Therefore after writing every block, the parity is calculated and written as the parity block. These tasks are CPU intensive, and thus by reducing the number of Portal RPC threads, they can use more CPU. But the Portal RPC threads take charge of communication between the OSS and client. Consequently, if we reduce the number of Portal RPC threads to too small a number, the communication becomes a performance bottleneck.

V. Discussion

It is well known that Lustre can perform I/O operations using a file stripe. Therefore the throughput is higher when the stripe count is greater than 2 [13]. But in our experiments mentioned above, the throughput dropped when the stripe count was greater than 1. We conjecture that the throughput drop can occur because Lustre performs block I/O operations in a synchronous manner when using direct I/O [12].

We assume that the reason why the results shown in Subsection IV-B were obtained is related to direct I/O. First, as just mentioned, I/O operations were performed synchronously when using direct I/O in Lustre. We used the direct I/O mode in running these benchmarks. Lustre cannot parallelize the synchronous operation, and therefore the use of too many Portal RPC threads is excessive. We achieved the best performance when the OSS had Portal RPC threads, the number of which was about 2/3 of that of system cores and a client had Portal RPC threads, the number of which was about 1/5 of that of system cores. Based on this result, we can conclude that the clients need more CPU powers to perform their workload (since they run in compute nodes) whereas OSSes utilize most of the CPU powers for performing Lustre operations.

VI. Conclusion

We compared the existing distributed parallel file systems which are widely being used in HPC environments, and

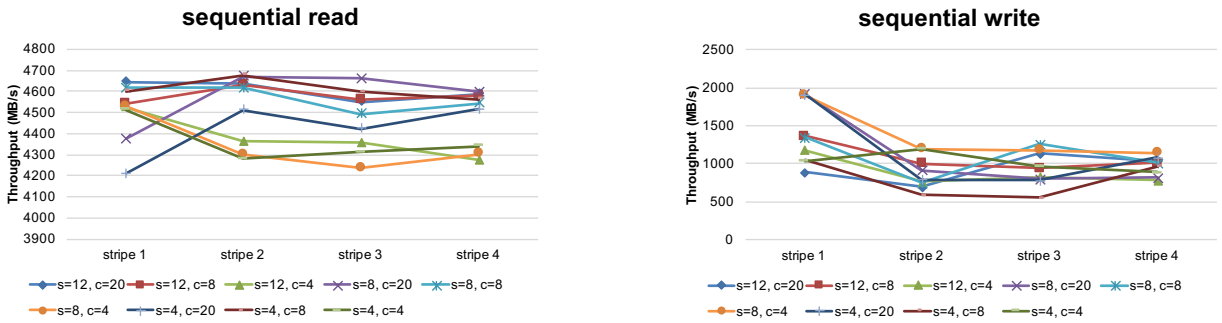

Fig. 2: Sequential I/O performance with 1 client, 16 threads, and 1GB block size

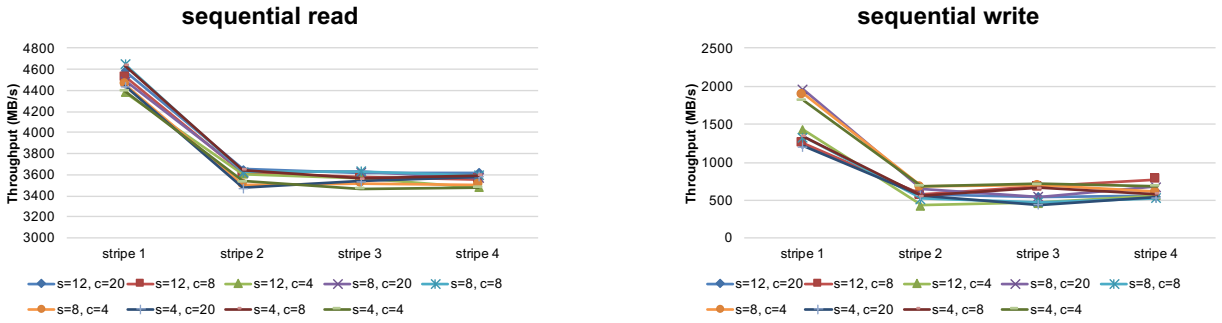

Fig. 3: Sequential I/O performance with 1 client, 8 threads, and 2GB block size

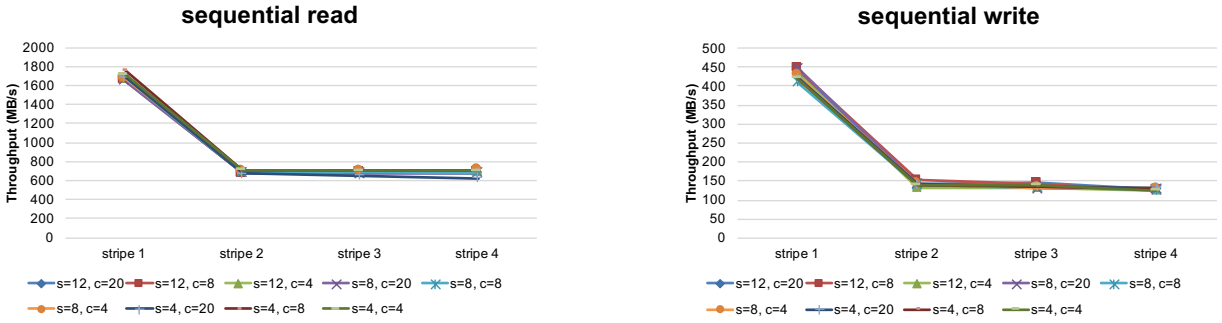

Fig. 4: Sequential I/O performance with 1 client, 1 thread, and 16GB block size

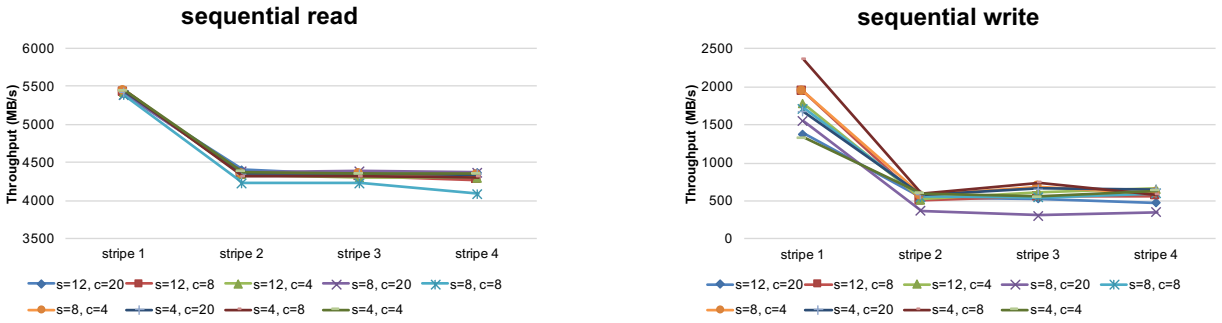

Fig. 5: Sequential I/O performance with 8 clients, each with 1 thread, and 2GB block size

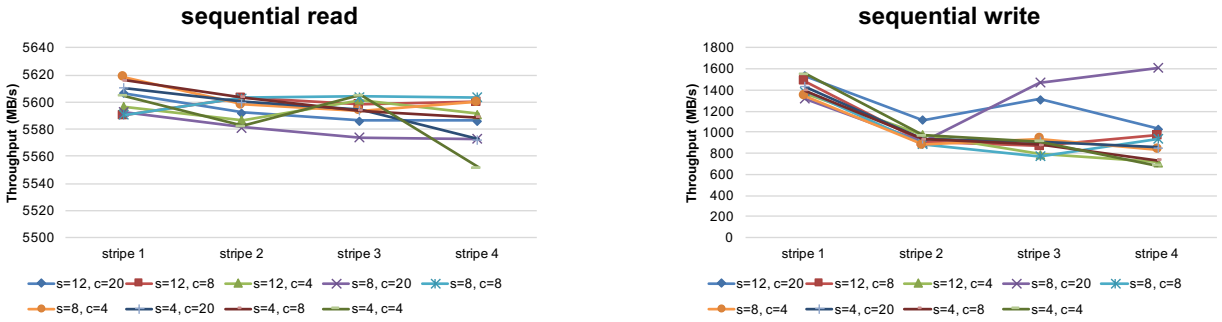

Fig. 6: Sequential I/O performance with 8 clients, each with 8 threads, and 256MB block size

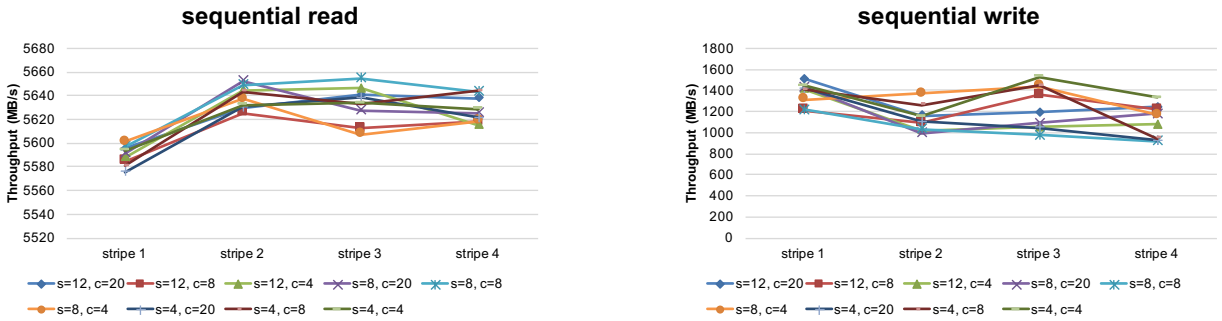

Fig. 7: Sequential I/O performance with 8 clients, each with 16 threads, and 128MB block size

investigated which parts can become performance bottlenecks. Then we analyzed the characteristics of the Lustre file system especially when using the direct I/O mode, attempting to identify such bottlenecks. We executed a popular benchmark on Lustre with various configurations and parameter settings, and found out the settings which lead to the best performance. Specifically, we found that we can obtain up to 114% performance improvements by adjusting the number of Portal RPC threads on OSSes and Clients. Based on our experimental results, we expect to achieve performance improvements for various HPC applications, since their I/O performance can be improved. In addition, we plan to perform research on optimizing distributed parallel file systems more effectively by using the results.

ACKNOWLEDGMENT

This Research was performed as a subproject of project No. K-15-L01-C01 and supported by the KOREA INSTITUTE of SCIENCE and TECHNOLOGY INFORMATION (KISTI). This work was also partly supported by the Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korean government (MSIP) (No. R0190-15-2012, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development) and Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2013R1A1A2064629). In addition, this work was supported by BK21 Plus for Pioneers in Innovative Computing(Dept. of Computer Science and Engineering, SNU) funded by National Research Foundation of Korea(NRF) (21A20151113068).

REFERENCES

[1] "Lustre," http://www.lustre.org/, [Online; accessed 16-May-2016].
[2] F. B. Schmuck and R. L. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters." in *FAST*, vol. 2, 2002, pp. 231–244.
[3] "OrangeFS," http://www.orangefs.org/, [Online; accessed 16-May-2016].
[4] T. Crowe, N. Lavender, and S. Simms, "Scalability testing of dne2 in lustre 2.7," http://cdn.opensfs.org/wp-content/uploads/2015/04/Scalability-Testing-of-DNE2-in-Lustre-27_Simms_V2.pdf, [Online; accessed 16-May-2016].
[5] R. Hedges, K. Fitzgerald, M. Gary, and D. M. Stearman, "Comparison of leading parallel nas file systems on commodity hardware," in *Petascale Data Storage Workshop*, vol. 2010, 2010.
[6] Y. Tanimura, R. Filgueira, I. Kojima, and M. Atkinson, "Mpi collective i/o based on advanced reservations to obtain performance guarantees from shared storage systems," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–5.
[7] S. Roberts, "Next generation storage for the hltcoe," 2013.
[8] K. Sakai, S. Sumimoto, and M. Kurokawa, "High-performance and highly reliable file system for the k computer," *FUJITSU Science Technology*, vol. 48, no. 3, pp. 302–209, 2012.
[9] S. Sumimoto, S. Matsui, K. Sakai, F. Sueyasu, F. Shoji, A. Uno, and K. Yamamoto, "Metadata access reduction of large scale lustre based file system," http://cdn.opensfs.org/wp-content/uploads/2015/04/Metadata-Access-Reduction-and-Analaysis_Yamamoto_Sumimoto.pdf, [Online; accessed 16-May-2016].
[10] M. Pershin, "Intel lustre* data on mdt/small file i/o," http://cdn.opensfs.org/wp-content/uploads/2014/04/D1_S10_LustreFeatureDetails_Pershin.pdf, [Online; accessed 31-May-2016].
[11] W. Loewe, T. McLarty, and C. Morrone, "Ior benchmark," 2012.
[12] F. Wang, S. Oral, G. Shipman, O. Drokin, T. Wang, and I. Huang, "Understanding lustre filesystem internals," *Oak Ridge National Laboratory, National Center for Computational Sciences, Tech. Rep*, 2009.
[13] S. C. Simms, G. G. Pike, and D. Balog, "Wide area filesystem performance using lustre on the teragrid," in *TeraGrid Conference*. Citeseer, 2007.