PERFORMANCE OF SMALL BLOCK SIZE ACCESSES IN LUSTRE FILE SYSTEM

[1]Anuja Kulkarni, [2]Dr. Naveenkumar Jayakumar
[1]M.Tech Scholar, [2]Associate Professor
[1]Department of Computer Engineering,
[1]Bharati Vidyapeeth Deemed University College of Engineering, Pune, India

Abstract— In big data environment, most of the files (70%) are small, and most data (nearly 90%) is placed in big file. The number of small files is big though used space is not. Small files consume more resources and produce big slowdown. Also, the latency of access to small files is important. The problem is studied by considering Lustre file system as a use case because the Lustre file system is open source, most widely used in high performance computing (HPC) environment and easy to implement file system.

IndexTerms— File system, Lustre, Big data, Small files

I. INTRODUCTION

Lustre is a parallel file system used in a wide range of HPC environments. Lustre file system is the most widely used by the world's top 500 HPC sites. [20]

The main advantage of Lustre file system over Storage Area Network (SAN) file system and Network File System (NFS) is that it provides: a global name space, the ability to distribute very large files across multiple storage nodes, wide scalability in performance as well as storage capacity. Since large files are distributed across many nodes in the cluster environment, Lustre file system is best suited for high-end HPC cluster I/O systems.

Lustre is an open source file system which is used primarily for Linux-based HPC clusters. It is implemented using few server nodes and client nodes called as Lustre clients.

The Lustre file system offers scalability due to which it is a popular choice in the finance, manufacturing and oil and gas sectors. The scalability offered by Lustre reduces the common requirement of creating many separate file systems, such as one for each cluster. This Lustre feature provides significant storage management.

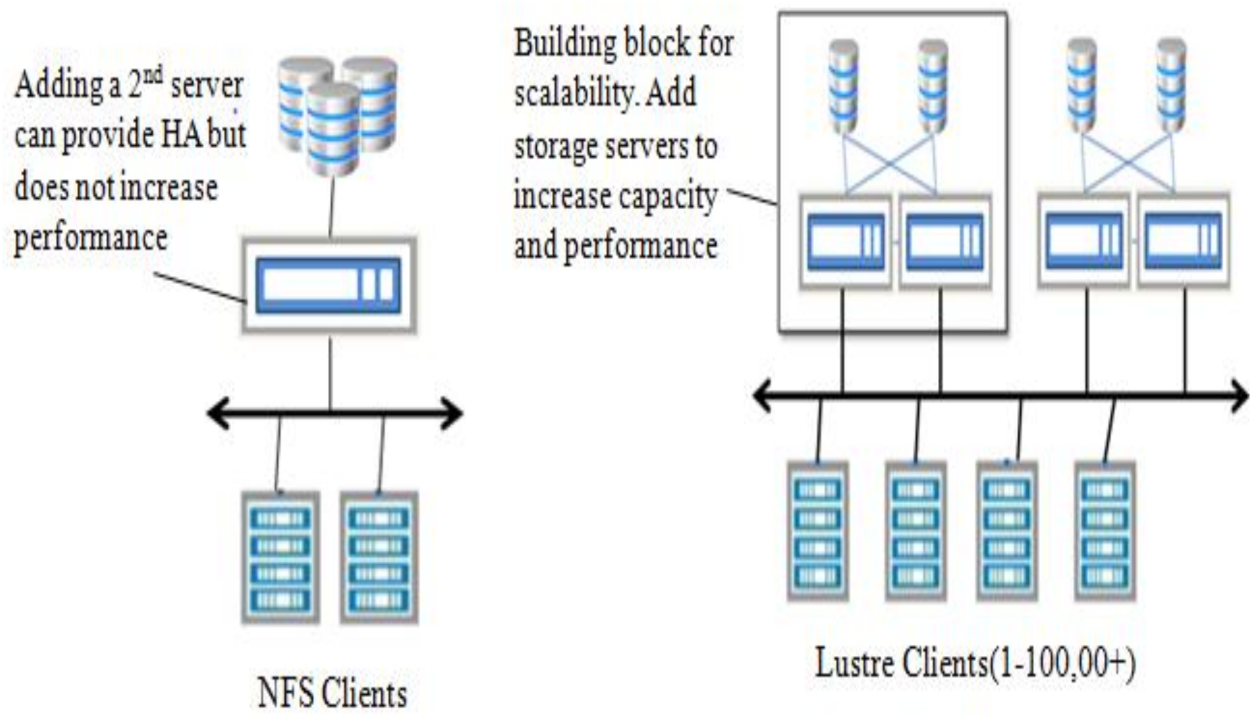
Adding a 2nd server can provide HA but does not increase performance
NFS Clients
Building block for scalability. Add storage servers to increase capacity and performance
Lustre Clients(1-100,00+)

Fig. 1: Traditional Network File System vs Lustre File System

Lustre servers are equipped with multiple storage devices which provide high-availability. Lustre can handle and serve up to tens of thousands of clients. The high-availability mechanism should enable any cluster file system to handle server failures or reboots transparently. The Lustre failover mechanism is robust and transparent, and it allows servers to be upgraded without the need to take the system down.

Lustre file system is designed for large sequential streaming I/O. This generally impacts the performance with small files. [2] The data stored in Lustre file system is striped across multiple OSTs (Object Storage Target), but the layout of the stripe is generally not optimized for the small files. Many of the small files could impact the performance of single OST where they get stored and also slow down other workloads on the file system.

II. BACKGROUND

Lustre Architecture

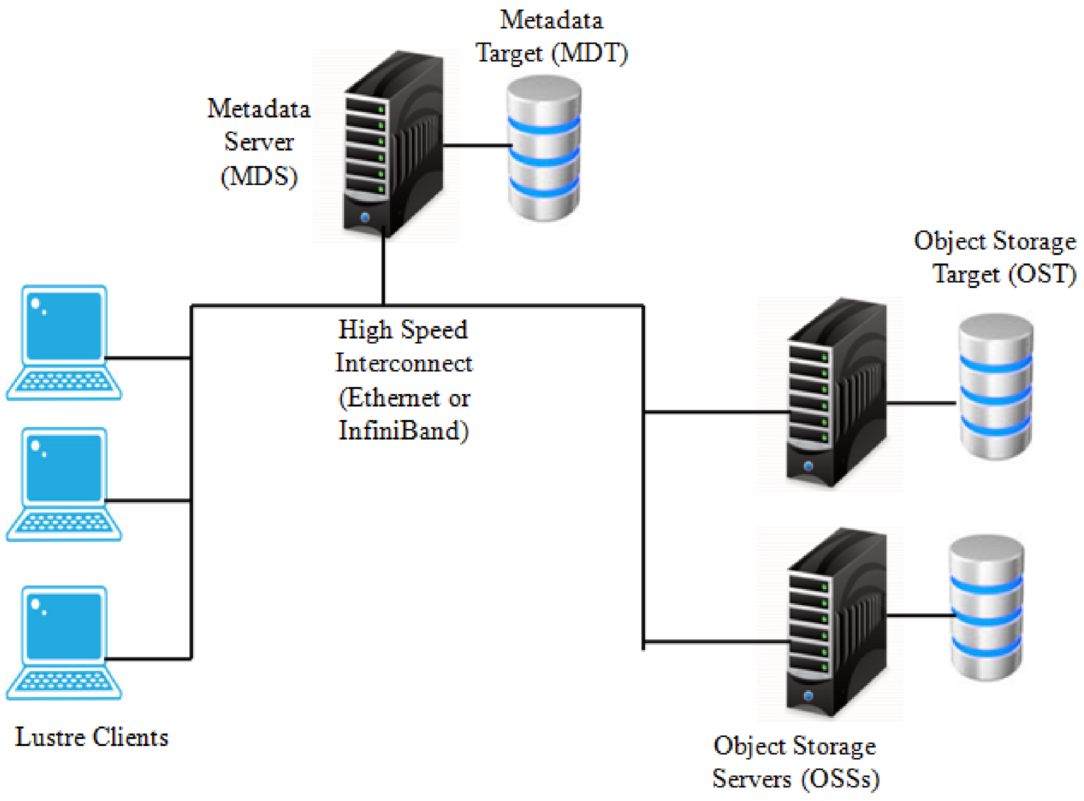
Metadata Target (MDT)
Metadata Server (MDS)
Object Storage Target (OST)
High Speed Interconnect (Ethernet or InfiniBand)
Lustre Clients
Object Storage Servers (OSSs)

Fig. 2: Lustre Components

Lustre is an object-based file system with three main components: Object Storage Servers (OSSs), Metadata Servers (MDSs), and clients.[6] Lustre components are as shown in the fig. 2.

For storing file data and metadata, Lustre uses block devices which are managed by Lustre services. Lustre clients access the data through standard POSIX I/O system calls.

Metadata server (MDS) provides metadata services. Correspondingly the Metadata Client (MDC) is a client of those services that makes metadata available to the Lustre clients. File metadata, such as file names, access permissions, directory structures, is stored on the Metadata Target (MDT).The management server (MGS), stores configuration information for all available Lustre file systems in a cluster. Lustre clients and Lustre target contacts the MGS to retrieve and provide information respectively. The MGS can have a separate disk for storage, or it can share a disk with a single MDT. The OSS exposes block devices and serves data to the client. Correspondingly, Object Storage Client (OSC) is a client of the services. Each OSS manages one or more OSTs (Object Storage Targets). OSTs are used to store file data in the form of objects.

Working of Lustre File System

In Lustre, file operations like create, open, write, read, etc. require metadata information which is stored on MDS. This service provided by MDS is accessed through client interface module, known as Metadata Client (MDC).
From the MDS point of view, every single file is composed of numerous data objects, and these objects are striped across one or more OSTs. Each data object is assigned with unique object id. MDS stores normal file metadata attributes like inode with some additional information known as Extended Attributes (EA).[17]

Before reading the file stored on the OSTs, client will communicate with MDS via MDC (Metadata Client) and collect the information about OSTs where the objects of the file are stored.
Now the client can communicate with corresponding OSTs through a client module interface known as OSC (Object Storage Client).

In Lustre, the communication between client and server is coded as an RPC (Remote Procedure Call) request and response. This middle layer is known as ptl-rpc i.e., Portal RPC which translates the file system request in the form of RPC request and response and the Lustre Networking(LNET) provides network infrastructure to put that down onto the wire.

Small File Performance in Lustre

The read/write performance of Lustre file system is currently optimized for large files that are the files more than few megabytes in size. To access any file client has to send initial open RPC to the MDT and after that, to fetch the data from the OSTs, there are separate read/write RPCs to the OSTs. In addition to this, there are separate RPCs to perform disk I/O on MDT and OST. This functionality separation is desirable for large files (more than few megabytes in size) since one open RPC to MDT requires less execution time compared to total number of read/write RPCs to OSTs, but this affects overall performance of small file significantly when there is only one read or write RPC to the OSTs for accessing file data.

A set of IOzone [4] test is executed on the client to find out the influence of record size. All the tests are performed based on synthetic data generated by IOzone. To evaluate the read/write performance with IOzone, the file size is set to 2G with stripe size 4M and data striped across 2 OSTs. The figure shows the results of the tests. The vertical axis is the read/write bandwidth (data transfer rate in bits/sec), and the horizontal axis is the record size. The red line shows write performance, and the blue line shows read performance.

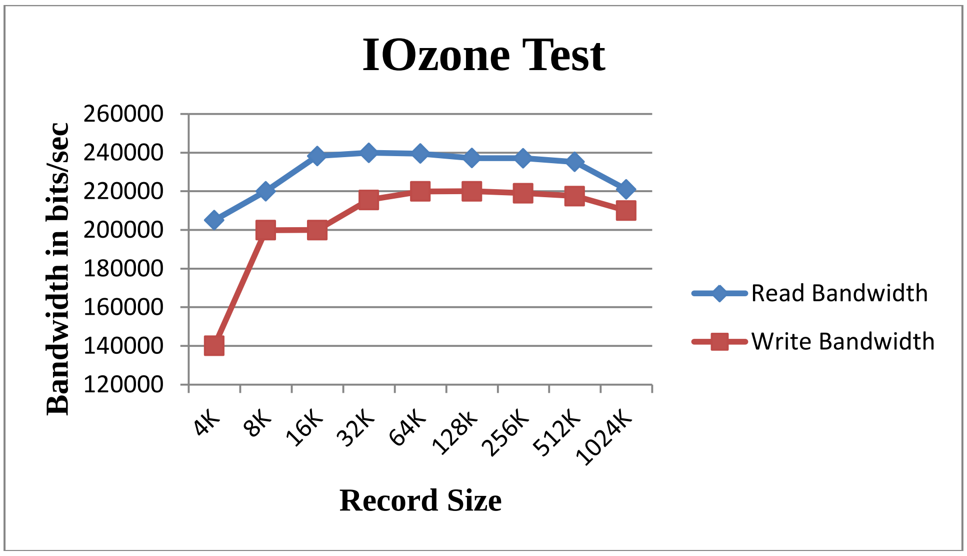

Fig. 3: Benchmarking with IOzone

As shown in the above graph fig. 3, the trend of the two lines is almost the same, but when the record size is 128K, 256K, and 1M, the performance is better than others. For the record size 4K or 8K, most of the RPC packets contain only one or two pages and hence the I/O efficiency is much low. From the graph above, it can be concluded that for small record size less number of RPCs are required and the overall I/O performance get affected when there is only one read or write RPC to the OSTs for accessing the file data.

III. PROPOSED SOLUTION

The research aims to improve the performance of small files by putting small files data only on MDT so that the additional read/write RPCs to the OSTs can be eliminated. This allows improving the small file performance. The MDT storage is configured with RAID 1+0 which is well-known for high-IOPS. Data on MDT can be used in conjunction with DNE (Distributed Namespace) to improve the efficiency.

To store file data on the MDT, system administrators or users must explicitly specify a layout that will allow storing the data on MDT at the time of file creation.

The maximum file size for which data can be stored on MDT must be specified by the administrator so that users cannot store large files on MDT which will cause the problems to other users. If the layout of a file specifies to the client to store the data on the MDT, but the file size reaches to the maximum size specified by the administrator, then the data will be migrated to OST.

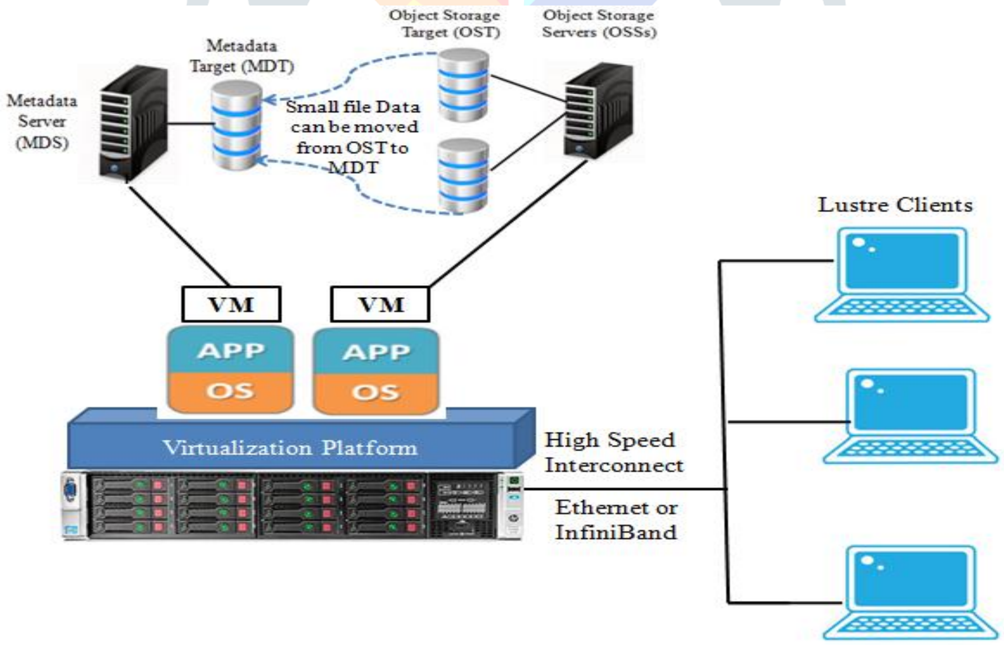

Fig. 4: System Architecture

The figure 4 shows proposed system architecture in which two virtual machines are created. Here, both the virtual machines are formatted with CentOS 6.7 operating system and kernel is patched with Lustre 2.7.0 software release.
One virtual machine is configured with MDS (Metadata Server) having single MDT (Metadata Target) and other with OSS (Object Storage Server) with two OSTs (Object Storage Target).

Lustre clients are connected to the server over Ethernet connection, and they are also formatted with CentOS6.7 operating system.

In native Lustre architecture, the small file data is present on the OST. Whenever the client wants to access data from small files, it has to send RPCs to both MDS and OSS. As the number of RPCs get increased the overall latency increases which affect the I/O performance.

In the proposed system, the data for small files will be stored on MDT instead of OSTs. Since the data and metadata both are present on MDT, the client has to send RPC to MDS only. As the number of RPCs decreases the overall latency also decreases and I/O performance for the small files can be improved to some extent.

IV. CONCLUSION

The read/write performance of Lustre file system is currently optimized for large files that are the files more than few megabytes in size. The read/write performance depends on the number of LOOKUP RPCs between client to OST and client to MDT. One possible way to improve small file performance in Lustre is to put the data for small files only on the MDT where metadata also resides so that additional RPCs and I/O overhead can be eliminated. In the proposed research work the small file performance has been tested using IOzone benchmarking tool. The future work for the proposed scheme is to check whether it is feasible and beneficial to move the small files to the MDT so that additional RPCs and I/O overhead can be eliminated, and read/write performance of Lustre file system can be improved.

REFERENCES

[1] Acharya, A., Uysal, M., & Saltz, J. (1998). Active disks: programming model, algorithms and evaluation. *SIGPLAN Not.*, *33*(11), 81–91. http://doi.org/10.1145/291006.291026
[2] Adam Roe – HPC Solutions Architect HPDD Technical Consulting Engineering. (n.d.).
[3] Brandt, S. A., Miller, E. L., Long, D. D. E., & Xue, L. (2003). Efficient metadata management in large distributed storage systems. *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST)*, (April).
[4] Centre, T. S. (2008). Study of the Lustre file system performances before its installation in a computing Cluster.
[5] Chipset, A. W. L. a N., & Meng, T. H. (2003). Design and Implementation of an, *50*(August), 160–168.
[6] Cluster File Systems Inc. (n.d.). Lustre: A Scalable, High-Performance File System. *Lustre*, 1–13.
[7] Data-on-mdt, T., & Architecture, D. S. (n.d.). Data On MDT High Level Design Make MDC devices part of CLIO.
[8] Data, T., Mdt, O., Raid-, I., & Namespace, D. (n.d.). Data On MDT Solution Architecture Design a new layout for files with data on MDT .
[9] Depardon, B., Mahec, G. Le, & S´eguin, C. (2013). Analysis of Six Distributed File Systems, 44. Retrieved from https://hal.archives-ouvertes.fr/file/index/docid/789086/filename/a_survey_of_dfs.pdf
[10] Deshmukh, S. C., & Deshmukh, P. S. S. (2015). Simple Application of GlusterFs : Distributed file system for Academics, *6*(3), 2972–2974.
[11] Dhawan, U., Hritcu, C., Rubin, R., Vasilakis, N., Chiricescu, S., Smith, J. M., … DeHon, A. (2015). Architectural Support for Software-Defined Metadata Processing. *ASPLOS '15: Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 487–502. http://doi.org/10.1145/2694344.2694383
[12] Dhawan, U., Vasilakis, N., Rubin, R., Chiricescu, S., Smith, J. M., Knight Jr, T. F., … DeHon, A. (2014). PUMP: a programmable unit for metadata processing. *Workshop on Hardware and Architectural Support for Security and Privacy*, 8. http://doi.org/10.1145/2611765.2611773
[13] Felix, E., Fox, K., Regimbal, K., & Nieplocha, J. (2006). Active storage processing in a parallel file system. *Proc. of the 6th LCI …*. Retrieved from http://hpc.pnnl.gov/active-storage/papers/lci-evanfelix.pdf
[14] Fu, Y. F. Y., Xiao, N. X. N., & Zhou, E. Z. E. (2008). A Novel Dynamic Metadata Management Scheme for Large Distributed Storage Systems. *2008 10th IEEE International Conference on High Performance Computing and Communications*, (September), 987–992. http://doi.org/10.1109/HPCC.2008.86
[15] Kim, W., & others. (2005). On metadata management technology: status and issues. *Journal of Object Technology*, *4*(2), 41–47. Retrieved from http://www.jot.fm/issues/issue_2005_03/column4/
[16] Kling-petersen, T. (n.d.). LUSTRE ^{TM} FILE SYSTEM :, (820).
[17] Kulkarni, A., & Jayakumar, N. (2016). A Survey on IN-SITU Metadata Processing in Big Data Environment, *9*(44), 325–330.
[18] Levy, E., & Silberschatz, A. (1990). Distributed file systems: concepts and examples. *ACM Computing Surveys*, *22*(4), 321–374. http://doi.org/10.1145/98163.98169
[19] Meshram, V., Ouyang, X., & Panda, D. K. (n.d.). Minimizing Lookup RPCs in Lustre File System using Metadata Delegation at Client Side.
[20] Paper, W. (n.d.). Application Performance for High Performance Computing Environments, 1–11.
[21] Piernas, J., Nieplocha, J., & Felix, E. J. (2007). Evaluation of active storage strategies for the lustre parallel file system. *Proceedings of the 2007 ACMIEEE Conference on Supercomputing SC 07*, (1), 1. http://doi.org/10.1145/1362622.1362660
[22] Ren, K., Zheng, Q., Patil, S., & Gibson, G. (2014). IndexFS : Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion.
[23] S.~A.~Weil, S.~A.~Brandt, E.~L.~Miller, D.~D.~E.~Long, & C.~Maltzahn. (2006). {Ceph}: A scalable, high-performance distributed file system. *Proceedings of the 7th Symposium on Operating Systems Design and Implementation ({OSDI})*, 307–320.
[24] Wang, F., Oral, S., Shipman, G., Drokin, O., Wang, T., & Huang, I. (2009). *Understanding Lustre Filesystem Internals*. *Ornl/Tm-2009/117*. Retrieved from http://wiki.lustre.org/images/d/da/Understanding_Lustre_Filesystem_Internals.pdf
[25] Weil, S., Pollack, K., Brandt, S., & Miller, E. (2004). Dynamic metadata management for petabyte-scale file systems. *Proceedings of the 2004 …*, *0*(c), 4–4. http://doi.org/10.1109/SC.2004.22
[26] Welch, B., Unangst, M., Abbasi, Z., Gibson, G., Mueller, B., Small, J., … Zhou, B. (2010). White Paper Scalable Performance of the Panasas Parallel File System, (May), 1–22.
[27] Zhang, G., Shu, J., Xue, W., & Zheng, W. (2007). Design and implementation of an out-of-band virtualization system for large SANs. *IEEE Transactions on Computers*, *56*(12), 1654–1665. http://doi.org/10.1109/TC.2007.70765