A guideline for data placement in heterogeneous distributed storage systems

Shun Kaneko, Takaki Nakamura
Research Institute of Electrical Communication
Tohoku University
Sendai, Japan

Hitoshi Kamei
Research and Development Group
Hitachi, Ltd.
Yokohama, Japan

Hiroaki Muraoka
Research Institute of Electrical Communication
Tohoku University
Sendai, Japan

***Abstract*—We propose a guideline for data placement in heterogeneous distributed storage systems. Heterogeneous distributed storage systems sometimes cause degradation of aggregate data throughput. The proposed guideline is that data accessed by a client should be placed equally to all servers. We evaluate a storage system configured by the proposed guideline by using sysstat and compare read/write data throughput between typical data placement and proposed data placement. Then we conclude that the proposed guideline improves the rate of aggregate data throughput while increasing the number of access streams.**

Keywords—distributed storage system; data placement; data throughput; GlusterFS

I. Introduction

Many kinds of distributed storage systems have been proposed to improve aggregate data throughput [1][2]. Most of the distributed storage systems take a scale out architecture, which allows them to add new storage nodes (clients and servers) to an existing system. The nodes of the distributed storage system sometimes becomes heterogeneous which means different specifications.

The first use case is a system expansion. In general, specifications of new storage nodes for the expansion may be different from storage nodes used in the existing system. Because computer technologies may evolve during the long time from the day of system installation to the day of system expansion. Therefore, aged distributed storage systems tend to become heterogeneous.

The second use case is the reuse of computer resources. When a storage administrator composes a distributed storage system of unused computer resources which were used for another systems before. In this case, it must be difficult to collect many computers with same specifications. Therefore, reused distributed storage systems tend to become heterogeneous.

These heterogeneous distributed storage systems sometimes fail to keep balanced load between the nodes. In this paper, we show a typical case of performance degradation on the heterogeneous distributed storage systems. Then we propose an appropriate data placement guideline for the heterogeneous system. We compare the results of sysstat and read/write data throughput.

This work is supported as "Research and Development on Highly Functional and Highly Available Information Storage Technology," sponsored by the Ministry of Education, Culture, Sports, Science and Technology in Japan.

The remainder of this paper is organized as follows. We explain GlusterFS as one of distributed storage systems and the issue of heterogeneous GlusterFS environment in section 2. In section 3, we propose the data placement guideline for the heterogeneous systems. In section 4, we compare the results of sysstat and read/write data throughput between typical data placement and proposed data placement. We present related work and conclusions in section 5 and 6, respectively.

II. Overview of GlusterFS and the Issue

A. Overview of GlusterFS

GlusterFS [1] is a distributed file system which supports not only the gluster native protocol but also de-facto standard protocols such as NFS and CIFS. Unlike conventional distributed file systems, GlusterFS doesn't have a central server such as metadata server. GlusterFS is implemented on FUSE which is a framework to implement filesystems on user space. As GlusterFS is open source software, it can be used on various platforms, including Linux.

Figure 1 illustrates steps of data access for a file stored on GlusterFS. A GlusterFS system consists of multiple clients and storage servers. For simplicity in this paper, each storage server has one brick created on one disk although it can have multiple disks and multiple bricks can be created on one disk. When an application creates a file, a client of GlusterFS generates a hash value based on the file name. As the same hash algorithm is

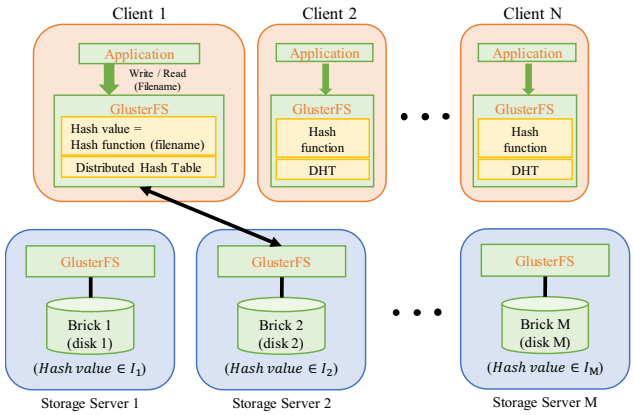

Fig.1. Steps of data access on GlusterFS

978-1-4673-8985-3/16 $31.00 © 2016 IEEE
DOI 10.1109/IIAI-AAI.2016.162

commonly used among clients, the hash value of the file is determinately fixed once the file name is fixed. Each storage server of GlusterFS is assigned a range of hash values based on a distributed hash table. Therefore, the storage server for the file is also determinately fixed without a central server. When an application reads a file, the system of GlusterFS works in the similar manner. The hash value is calculated from the file name for reading the file. Then clients send read requests to the storage server corresponding to the hash value.

GlusterFS provides three types of volume: a distributed volume, a replicated volume, and a striped volume. The volume consists of multiple bricks (same as disks in this environment.) The distributed volume stores a file in a disk of the volume. It is suitable for dealing with a large number of files. The replicated volume replicates a file to all disks composing the volume. It gives redundancy to the file. It has a disadvantage that disk utilization efficiency is low. The striped volume stores a file among disks composing the volume after the file is divided into the number of disks. It increases data throughput for a large file at the risk of lower reliability. We use the distributed volume in this paper as it is suitable for a general dataset consisting of multiple files..

B. *The Issue with GlusterFS*

GlusterFS has additional advantages that is flexible scalability, and the stability of the system. However, aggregate data throughput sometimes degrades when specification of nodes is heterogeneous. The nodes include storage servers and clients.

Figure 2 shows an example of typical data placement on GlusterFS. The system consists of four storage servers and four clients. Each storage server has six disks. One volume of GlusterFS is created for each client. Therefore, each storage server has one volume. A client accesses one specified storage server. For example, client1 accesses a volume consisting of disk1-1, disk1-2, disk1-3, disk1-4, disk1-5, and disk1-6 on server1.

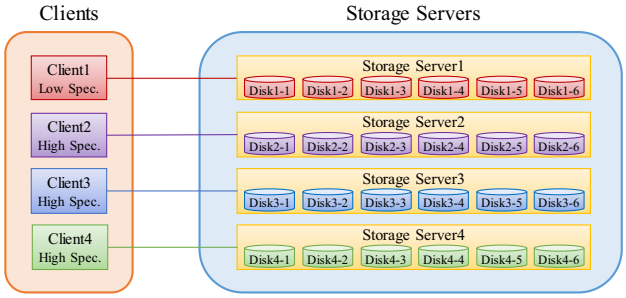

Fig.2. Typical data placement of GlusterFS

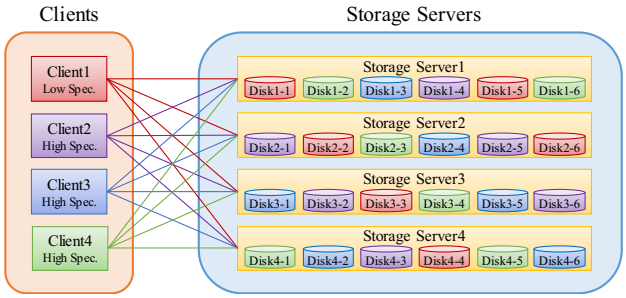

Fig.3. Data placement based on proposed guideline

In this configuration, if the specifications of client1 is lower than the other three clients, performance degradation may occur. This is because client2, client3, and client4 finish assigned data access tasks before finishing a task assigned to client1.

One possible solution for this issue is to rebalance the tasks in proportion to the specifications of nodes. However, it is difficult to quantify a performance of each node for a task based on the specifications. Moreover, the performance depends on the type of the task.

Therefore, a simple method or guideline of load balancing for the heterogeneous distributed storage system may be an improvement.

III. Proposed Guideline

In the typical data placement shown, storage server2, server3, and server4 may fall into idle state because client2, client3, and client4 finish the assigned task earlier than client1.

To avoid this underutilization, we propose a simple guideline for data placements so that a volume always consists of disks on no single storage server but all storage servers. The proposed guideline should prevent any server from falling into idle state.

Figure 3 shows the data placement based on the proposed guideline on the same system configuration as shown in Fig. 2. Each volume is created by a plurality of disks on all storage servers. For example, the volume for client 1 consists of disk1-1 on server1, disk2-2 on server2, disk3-3 on server3, disk4-4 on server4, disk1-5 on server1, and disk2-6 on server2.

IV. Evaluation

A. *Environment of Evaluation*

For evaluations, we use four storage servers and four clients. Clients consist of one node with low specifications and three nodes with high specifications. Table 1 shows the low specifications of the client. Table 2 shows the high specifications of the clients. Storage servers consist of four nodes with same specifications. Table 3 shows the specifications of storage

TABLE I. Client Specifications (low)

Device	Specifications
CPU	2.67GHz 4Cores Xeon E5640
Main Memory	512MB

TABLE II. Client Specifications (high)

Device	Specifications
CPU	2.40z 8Cores Xeon E5-2630
Main Memory	512MB

TABLE III. Storage Server Specifications

Device	Specifications
CPU	2.20z 6Cores Xeon E5-2430
Main Memory	512MB

Device	Specifications
Disk (for data)	Seagate ST3600057SS 300GB×6 Drive interface: SAS (6 Gb/s) Rotating speed: 10000rpm Average latency: 2.90ms Buffer: 64MB Outer sustained transfer rate: 204MB/s. Inner sustained transfer rate: 125MB/s

servers. Each storage server has 6 disks for data. Although each node has physical memory more than 8 GB, we limit the memory size for OS to 512 MB to reduce the influence of cache. We use Linux kernel version 2.6, CentOS version 6.4, ext4 for disk format and GlusterFS version 3.3.1.

We use two network switches: one for clients and the other for storage servers. Each client is connected to the client-side switch via 10 Gigabit Ethernet. Each server is also connected to the server-side switch via 10 Gigabit Ethernet. Two switches are connected by link aggregation. In this case, the wire speed between two switches is 20 Gbps.

We measure write throughput and read throughput while accessing 10 GB sized files with changing the multiplicity. The files are located on servers specified by DHT in Fig,1. We set the multiplicity to 4, 8, 12, 16, 20, and 24. For example, when the multiplicity is 24, each client accesses six 10 GB sized files on six disks simultaneously. Therefore, 24 files are accessed by all clients at the same time.

B. *Analysis by Sysstat*

Sysstat is a package tool that can monitor CPU usage, memory usage, disk I/O, network I/O, and so on. It can save the resource states at fixed intervals and can view the time-series of resources states after experiments. We collect the time-series of network throughputs on storage servers at every 10 seconds in this experiment.

Figure 4 and Figure 5 show stacked charts indicating network throughput based on receiving and transmitting speed at the storage servers by collecting sysstat. The x-axis shows elapsed time from the starting time of measurement and the y-axis shows throughputs at every 10 seconds. We can see six blocks in the graph corresponding to the multiplicities of 4, 8, 12, 16, 20, and 24.

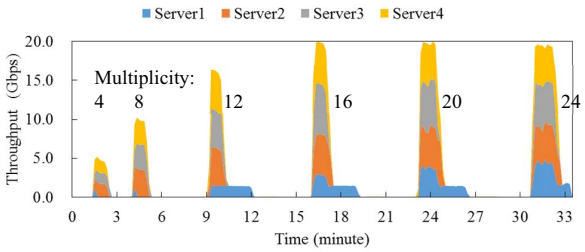

Fig.4. Read throughputs at every time-ticks on servers by sysstat on typical data placement

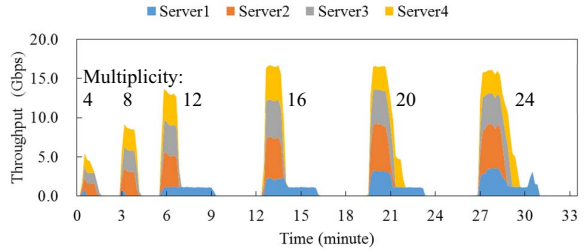

Fig.5. Write throughputs at every time-ticks on servers by sysstat on typical data placement

The finishing time of data transfer at server1 is later than the other servers when the multiplicity is equal to or more than 12. As the hardware specifications of server1 is the same as the other servers, this delay must be caused by client1's low specification status. In the read case, the peak value of aggregate throughput reaches 20Gbps which is equal to wire-speed between network switches when the multiplicity is equal to or more than 16. However, in the write case, the peak value of aggregate throughput does not reach 20Gbps and it is around 17Gbps.

C. *Aggregate Read Throughput*

In the reading measurement, each client reads 10 GB sized files on the disks at the same time after the files are created. To read the file we use dd command with following form:

dd if=*filename* of=/dev/null bs=1M count=10240

This form means that the command reads 1MB repeatedly till the total read size becomes 10 GB. The data read from the file is dumped into /dev/null. Each client executes more than one process of the dd command. The number of process equals to the value of M divided by 4, where M is the number of multiplicity.

Figure 6 shows the measurement result of aggregate read throughput with changing the multiplicity under the typical data placement and the proposed data placement. Each mark shows

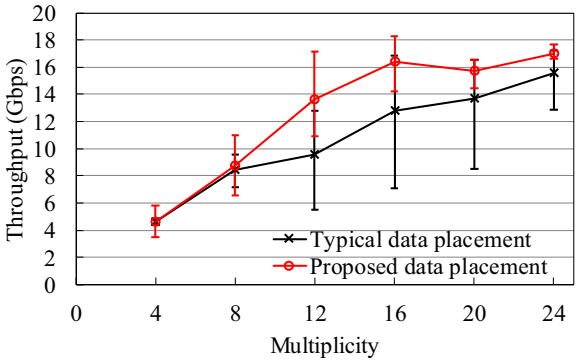

Fig.6. Aggregate read throughput

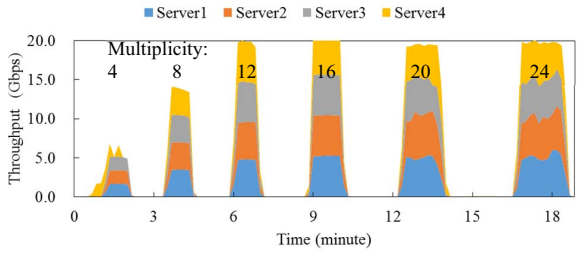

Fig.7. Read throughputs at every time-tick on servers by sysstat on proposed data placement

the average of three times measurement results. The error bar shows maximum value and minimum value of three times measurement results. Improvement of aggregate read throughput is observed when the multiplicity is equal to or more than 12. The improvement ratio in the range is from 9.4% to 42%, and the average of improvement ratio is 23%.

Figure 7 shows a stacked chart on proposed data placement. Unlike Fig. 4, the finishing times of data transfer at all servers are very close. Because of this, the aggregate read throughput is improved.

D. *Aggregate Write Throughput*

In the writing measurement, each client writes 10 GB sized files on the disk at the same time. To write the file we also use dd command with following form:

dd if=/dev/zero of=*filename* bs=1M count=10240

This form means that the command write 1MB repeatedly till the total write size becomes 10 GB. The data written in the file is zero. Each client execute more than one process of dd command as is the case with the reading measurement.

Figure 8 shows the measurement result of aggregate write throughput with changing the multiplicity under the typical data placement and the proposed data placement. Each mark shows the average of three times measurement results. The error bar shows maximum value and minimum value of three times measurement results. Improvement of aggregate write throughput is also observed when the multiplicity is equal to or more than 12. The improvement ratio in the range is from 31% to 47%, and the average of improvement ratio is 38%.

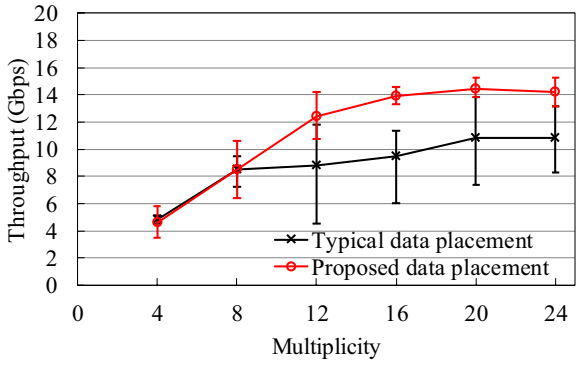

Fig.8. Aggregate write throughput

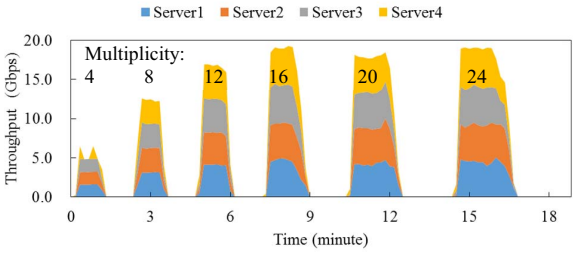

Fig.9. Write throughputs at every time-tick on servers by sysstat on proposed data placement

Figure 9 shows a stacked chart on proposed data placement. Unlike Fig. 5, the finishing times of data transfer at all servers are very close. Moreover, the peak value of aggregate throughput is also improved very well. Therefore, we think the write throughput can be improved even when we use a homogeneous environment. We assume two reasons. The first reason is that a client has a short request queue for one server. The second reason is that a storage server has a small number of service processes for one client.

V. Related Work

In the high-performance computing (HPC) environment, there are many distributed file systems that support to cluster volume servers. Generally, they do not accept the heterogeneous specs of the nodes.

Lustre [2] is used by super computers. The system consists of nodes which are clients, meta-data servers and the object storage servers. To access an object, the clients request the address of an object server storing the target object to the meta-data server, and they retrieve the object by the address information. Therefore, it is easy for the system to expand capacity and performance by adding the new object storage servers. The system does not specify the hardware specs of nodes; however, the system assumes the hardware specs are same configuration because of avoiding the I/O performance imbalance.

Information systems using P2P with DHT sometimes needs to solve load imbalance, thus many algorithms to address the problem are proposed. In that research area, studies focus mainly on an address space re-balancing of DHT.

Karger et.al proposes load-balance protocols for consistent hashing [3]. These protocols solve an imbalance of the address space and distribution of items among nodes. The protocols are useful for prevention of problems caused by DHT; however, it remains the data placement problems caused by heterogeneous hardware configuration.

VI. Conclusion

We propose a guideline of data placement for heterogeneous distributed storage systems. The proposed guideline is that data accessed by clients should be placed equally to all servers. We analyzed a performance bottleneck of a typical data placement. We measured read/write throughput on the typical data placement as well as the proposed data placement. Then we confirmed the improvement of aggregate read/write throughput.

It is important to extend the proposed guideline for striped volumes of GlusterFS, homogeneous environments, and the other distributed filesystems. These remain as a subject for future work.

References

[1] Developers, GlusterFS. http://redhatstorage.redhat.com/products/glusterfs/

[2] " Lustre Software Release 2.x Operations Manual," http://doc.lustre.org/lustre_manual.pdf

[3] David R. Karger and Matthias Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," SPAA '04 Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures, pp. 36-43, 2004.