Data Management on Grid Filesystem for Data-Intensive Computing

Hitoshi Sato
Tokyo Institute of Technology
hitoshi.sato@is.titech.ac.jp

Satoshi Matsuoka
Tokyo Institute of Technology
matsu@is.titech.ac.jp

Abstract

In parallel computing environments such as HPC clusters and the Grid, data-intensive applications involve large overhead costs due to a concentration of access to the files on common nodes. To avoid this problem in traditional distributed filesystems, users have to distribute the file access manually. However, such solution has some difficulties for users in the Grid environment. We propose a data management mechanism for data-intensive computing on Grid filesystem. Our technique improves the file access performance by automatically scheduling the file access and the data management on the filesystem. The filesystem is based on dynamically configured node groups corresponding to the network topology. Utilizing the configuration, it monitors file access to detect concentrated situations, creates the file replica, and schedules its placement and access. We applied the proposal technique to the Gfarm, a filesystem that scales to the Grid. We emulate real application workloads using a job scheduler and confirmed a speedup of factor 3.7 compared with a filesystem without automatic file access distribution techniques.

1. Introduction

Data-intensive applications on the Grid, such as high-energy physics, astronomy, life-science, etc., require to share and process a large amount of data generated by experiments or observations among multi-sites. In the environments, network shared filesystems, such as NFS, AFS, and cluster filesystems[4][3][5], are widely used to provide data accessibility and single system image, but there are some difficulties in using traditional distributed filesystems on the Grid . Some data-intensive applications involve large overhead costs due to a concentration of access to the files on common nodes. To avoid this problem in exsisting distributed filesystems, users have to distribute the file access manually. However, such solution is difficult for users in the Grid environment. To solve this problem, we propose a data management mechanism for data-intensive computing on Grid filesystem. Our technique improves the file access performance by automatically scheduling the file access and the data management on the filesystem. The filesystem is based on dynamically configured node groups corresponding to the network topology. Utilizing the configuration, it monitors file access to detect concentrated situations, creates the file replica, and schedules its placement and access. We applied the proposal technique to the Gfarm, a filesystem that scales to the Grid. We emulate real application workloads using a job scheduler and confirmed a speedup of factor 3.7 compared with a filesystem without automatic file access distribution techniques.

2. Problem of Filesystems on the Grid

There are a lot of advantages in the use of the network filesystem to federate resources on the Grid. It can provide not only data sharing among multi-sites but also single system image. The requirements of the filesystem on the Grid are mainly considered **Security** and **Scalability**.

NFS based filesystems combined with security mechanisms [6] support secure data sharing on a wide area network. However, these systems basically consist of a single node, causing possible performance bottleneck. Therefore, filesystems of this kind aren't suitable for the usage on the Grid in the point of scalability.

Striping parallel filesystems[4][3][5] are mainly used on HPC cluster environments to gain better I/O performance. All files are divided into fixed-size chunks, and each chunk can be placed in any storage node. However, the performance of these filesystems can often be limited by the network bandwidth, and most of these filesystems don't support security mechanisms that should satisfy the requirements on the Grid.

Grid Datafarm[7] is an architecture for petascale data intensive computing on the Grid. This system not only provides data sharing on the Grid but also schedules programs on nodes where the corresponding segments of data are stored to utilize local I/O scalability, rather than transferring the large-scale data to compute nodes. However, users need to manage data in the filesystem manually to improve

I/O performance in the heterogeneous resources. Moreover, even if one assume the use of the filesystems such as Grid Datarfarm, the situation that a concentration of file access from many clients to a single node and that access to file on remote sites may occur, which can be a performance bottleneck for applications running on the filesystems.

3. Data Management on the Grid Filesystem

To solve the problem described in Section 2, we propose a data management mechanism on grid filesystem to support data-intensive computing. Our main target application is the one that has write once and read only workload. We applied our proposal to Gfarm[2], which is a reference implementation of Grid Datafarm Architecture. Currently, our system is based on the version 1.2 of Gfarm.

3.1. Overview of the Filesystem

Figure 1 shows the overview of our target filesystem. The filesystem consists of three components: "client"(Gfarm Parallel I/O API), which offers interfaces for the filesystem, "metadata server"(gfmd and LDAP), which manages metadata of all files, and "I/O node"(gfsd), which maintains file fragments (We refer the "fragment" as "section"). First a client attempts to query the metadata server about a location of fragments of the file to which the client needs to access. This process is done by using Gfarm Parallel I/O API. Then, the metadata server selects an I/O node and notifies the node to the client. After that, the client access to the node. In addition , our technique uses a monitoring system on the metadata server(file system monitor(gfads)) and the I/O nodes(I/O node monitor(gfsd_am)). The filesystem monitor watches access to each file section on the filesystem to detect the concentration of access and the I/O node monitor watches resources on nodes (e.g. load, availability of the storage) to utilize them for scheduling file access and data management.

3.2. Management of nodes that compose the Filesystem

The nodes that compose the filesystem are divided into several groups based on the network topology. The aim of this division is to improve performance of file access to hide the heterogeneity of the Grid and to utilize the locality that has similarity among the node group. This grouping is done as follows. First we define *group size* s as below.

$$s(V) = \sum_{e \in E(V)} (\frac{1}{C_{bw}(e)} + C_{rtt}(e))$$

where V is a set of filesystem nodes that compose a group, E is a set of network links in the group V, C_{bw} is a network bandwidth, and C_{rtt} is a round trip time of the network. Based on this expression, *distance* d between two groups V_1 and V_2 are defined as follows.

$$d(V_1, V_2) = s(V_1 \cup V_2)$$

Using this expression, the filesystem nodes are diveded into several groups in the following steps: **(1)** Initialize groups that are composed of each filesystem node, **(2)** Calculate distance between groups using (1), **(3)** Select two groups whose distance are minimized and merge them into a single group, **(4)** Iterate the steps (2) and (3) until the minimum of the distance exceeds a threshold defined by the system previously. In the current implementation, the threshold is set at 0.01.

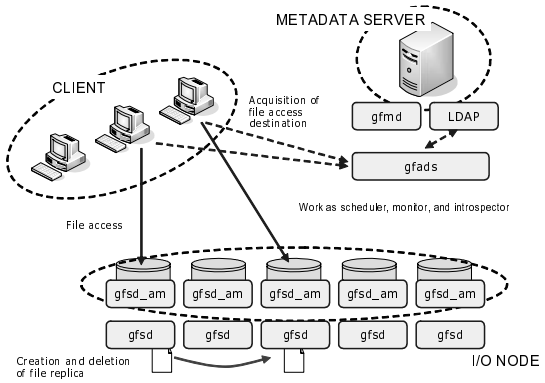

Figure 1. Overview of our proposal system

3.3. Control of file access from client

Under the node groups described in section 3.2, the metadata server schedules a file access destination from a client as follows: **(1)** If a target file **exists** on a local storage of a client, select the client node, **(2)** If a target file **doesn't exist** on a local storage of a client, and **(2-i)** if the target file **exists** on a node in a same group with the client, then select the node, or, **(2-ii)** if the target file **dosen't exist** on a node in a same group with a client, select any node that maintain the target file. If there exist several nodes that maintain the same file replica, the metadata server queries the nodes about load average and selects the node that advertize the minimum load average as the destination.

3.4. Data management on the Filesystem

Detection of concentration of filesystem node access The metadata server also monitors file access to detect a concentration of file access. This is conducted at each file section when the metadata server schedules an access destination. It records the tharget file and section name, the

client of the request, the scheduled I/O node, and the time. Using these information, our technique detects a concentration of file access as follows. First, the system defines an interval of monitoring time and a threshold for detecting a concentration previously. In the current implementation, the monitoring time is set at 180 sec. Next, it counts the file access during the interval from last file access. Then, we calculate a value as access state using following expression: $access\_state = \frac{count}{monitoring\_time}$. If the value of the access state is higher than the threshold, the metadata server decides the state as " access concentration " and attempts to replicate the file to another I/O node.

Replication of files When a concentration of file access is detected by the metadata server, the file section is replicated. The replication is done by calling Gfarm API (gfarm_url_section_replicate_to or gfarm_url_section_replicate_from_to) and the instruction is as follows: **(1)** If the scheduled node belongs to the **same** group that the client belongs to, replicate the file between **the group**, **(2)** If the scheduled node belongs to the **different** group that the client belongs to, replicate the file between **any two groups**. In other words, if the concentration occured by flash requests inside a group, the system increases the replica within it, and if the concentration occured by flash requests from different group, the system pulls the file near the group that the client belongs to.

Deletion of the file replicas When the metadata server doesn't detect a concentration of access to file section, replicas of the file are deleted according to the value of the access status. This process is conducted under the condition that the existence of the file is guaranteed.

4. Experiment

4.1. The Experimental Setups

To prove the validity of our proposal, We applied the data management mechanism to Gfarm and deployed a testbed, on which we executed a sample data-intensive program using a job scheduling system (Condor[1]). The focuses of the experiment are the effect of the detection of concentration and the data management. Figure 2 shows the configuration of the testbed, which consists of two separate clusters (PRESTO III located on Tokyo Institute of Technology (Titech) in Tokyo, and KOUME located on National Institute of Advanced Industrial Science and Technology (AIST) in Tsukuba). We deployed our prototype on this federated environment: A metadata server is allocated to one of the PRESTO III nodes, and clients and I/O nodes are allocated to other cluster nodes of PRESTOIII and KOUME.

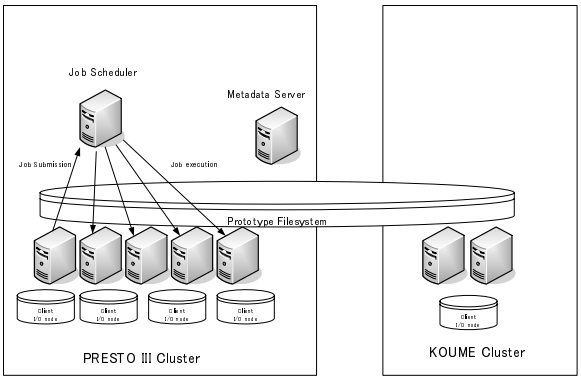

Figure 2. Configuration of the testbed

Table 1. Performance of each cluster

	PRESTO III	KOUME
num. of nodes	60 (120CPU)	5 (10CPU)
CPU	Opteron 242 ×2	Pentium III 1400MHz ×2
Memory	2GBytes	2GBytes
OS	Linux 2.4.27	Linux 2.4.20
Network	1000Base-T	100 Base-T

The specification of the cluster nodes and the network are shown in Table 1 and Table 2. We installed a Condor job scheduling system on PRESTO III and deployed one of the PRESTO III nodes as a job scheduler and other PRESTO III nodes as job submission hosts and job execution hosts.

On this environment, we submitted 100 jobs that open, read, and close files from one of the PRESTO III nodes to the job scheduler continually. The target files are placeed on the KOUME nodes. Therefore, the files are accessed from the job execution nodes via the filesystem. The file is composed of a single section and the size of the file is 128MB. We set the following experiment conditions: **(1)** ***norep***: The filesystem doesn't configure the nodes as groups nor manage the data, **(2)** ***same_config***: The filesystem configures all nodes as a single group statically and manage the data, **(3)** ***auto_config***: The filesystem configures the nodes as several groups dynamically using the methods described in section 3.2 and manage the data, **(4)** ***diff_config***: The filesystem configures the nodes as several groups statically and manage the data, **(5)** ***oracle***: The same condition as (1), but the data are distributed onto the local storage of all cluster nodes. In(4), We configure all nodes of KOUME as a group and the nodes of PRESTO III that is conncected to a single switch as a group. In the configuration (4), we compose 4 groups (7, 17, 16, 20 nodes) in PRESTO III. The parameter for the concentration detection is set at 0.1.

Table 2. Network performance

	PRESTO III nodes	KOUME nodes	PRESTO III - KOUME
rtt [msec.]	0.083	0.055	6.63
bandwidth [Mbits/sec]	973	908	73.0

Table 3. Average elapsed time [sec.]

	norep	same_ config	auto_ config	diff_ config	oracle
average	31.6	6.50	8.50	6.19	0.954

4.2. The Experimental Results

Table 3 shows average elapsed time of each condition and Figure 3 shows the relation between elapsed time of each job and accumulated number of the nonterminated jobs, which represents the behavior of the jobs. First, we compare "oracle" case with "no_rep" case. These results indicate that all jobs access the data on the local storage of their node on PRESTO III in "oracle" case because the elapse times of all jobs are the range of 0.8 - 1.0 sec, and that all jobs access the data on the storage of KOUME in "no_rep" case because they are over 29.8sec.

The effect of the node grouping are shown by the comparison of "same_config", "auto_config" and "diff_config" cases. In static node grouping methods, the different group configuration shows better performance than the same node group configuration. The result gives that static node grouping corresponding to the network topology can work effectively because access to files on remote sites can be reduced by the grouping. On the other hand, the dynamic node grouping method shows worse performance than other two methods for average elapsed time in Table3 and the time at 29.8sec in Figure3, which indicates that the method has overhead and doesn't work efficiently especially during initial state. However, Figure3 shows that the dynamic node grouping method shows better performance than other two methods in the range of 0.8 - 2.0 sec, which indicates that the dynamic node grouping method increases the localization of file access.

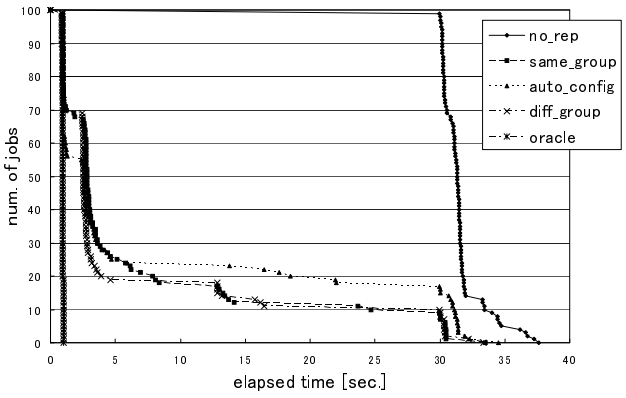

Figure 3. Behavior of each job in the exp.

Finally, we compare with "auto_config", "oracle", and "no_rep" cases. Our proposal shows 3.7 times speedup than the "no_rep" case. However, it shows 8.9 times speed down than the "oracle" case. The cause of the result is that the dynamic grouping method increases the average elapsed time by the maximam elapsed time of the jobs that access the data on KOUME. Therefore we need to further study the data management and transfer algorithms to utilize the locality on the Grid .

5. Conclusion

We propose a data management mechanism for data-intensive computing on Grid filesystem, which improves the file access performance by automatically scheduling the file access and introspecting the data placement. We applied the proposal technique to the Gfarm and emulate real application workloads using a job scheduler. We confirmed a speedup of factor 3.7 compared with a filesystem without automatic file access distribution techniques. Followings are the future work: (1) Further studies about the data management and transfer algorithms , (2) Evaluation on the large Grid environment with real applications.

.

References

[1] Condor project homepage. `http://www.cs.wisc.edu/condor`.

[2] Gfarm. `http://datafarm.apgrid.org/`.

[3] Lustre. `http://www.lustre.org`.

[4] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, 2000. USENIX Association.

[5] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 96–108, Bolton Landing, NewYork, October 2003. ACM Press.

[6] D. Maziéres. *Self-certifying File System*. PhD thesis, Massachusetts Institute of Technology, May 2000.

[7] O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi. Grid datafarm architecture for petascale data intensive computing. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 102–110, 2002.