International Journal of Control and Automation
Vol.8, No.11 (2015), pp.337-348
http://dx.doi.org/10.14257/ijca.2015.8.11.32

An Efficient Storage Technology for Object-Based Distributed Storage System

Jilin Zhang [abc], Yucheng Su [bc], Jian Wan [bc], Li Zhou [bc], Jue Wang[cd] and Lei Zhang[e]

a College of Electrical Engineering, Zhejiang University, Hangzhou, 310058, China
b School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, 310018, China
c Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, China
d Supercomputing Center of Computer Network Information Center, Chinese Academy of Sciences, Beijing, China
e Computer Science Department, Beijing University of Civil Engineering and Architecture Beijing, China

Abstract

There have two effective ways to improve the overall efficiency of the object-based distributed storage system, one is network caching technology and the other is metadata server cluster load balancing strategy. Under the heterogeneous storage environments, object-based distributed file system face the key challenge of lacking caching mechanism for the high-frequency access files, metadata server load imbalance also reduce the overall efficiency. To address these challenges, we discuss the design and implementation of an object-based distributed file storage caching system, and propose a weight-based metadata server cluster load balancing strategy. It effectively improve the metadata processing capacity. Based on the results, it demonstrated the feasibility and the efficiency of our methods.

Keywords: *Distributed File System, Object-Based Storage, Metadata Server Cluster, Storage Cache, Replacement Policy*

1. Introduction

Object-based distributed file system is the focus of current research in the storage field, the Object-based Storage System (OSS) is composed of the client interface, Object-based Storage Devices (OSDs) and Metadata Servers (MDSs), the metadata and data of the objects are managed separately in the OSS. OSD is responsible for the deployment and access to the underlying data blocks, maintain contact with the actual data blocks and logical data objects, and provide data objects' operator interface to upper client; MDS is an important part of the object-based storage system, play the role of the file system name space management, mapping between files and objects, user access control [1-4].

Under the heterogeneous storage environments, if the hot files are stored in a low-performance equipment, frequently access the object will appear high-latency phenomenon, this will reduce the overall storage efficiency and IO throughput. Most of object-based file system using pseudo-random function approximation of random assigned the data objects to each storage device. The MDSs with difference performance bear the same workload easily lead to weaker performance, become the system bottleneck, finally lead to a decline in overall system processing ability.

There are a lot of object-based distributed file systems use the network caching [5-7] mechanism to improve the performance of the system. Such as panFS [8] use of memory

ISSN: 2005-4297 IJCA

on a single OSD to cache data objects, but did not consider the data access features overall storage cluster. The cache mechanism of Lustre [9-10] save only a part of the data from the server on the client, the modified data will write back to the server at a suitable time, or to the OSD server prefect object dataset to cache for a read operation. This mechanism reduces network overhead during read and write operations when the client request for immediate data objects, but not for the overall storage cluster data access features to cache. Ceph [11] is approximate random to assign the data object to the OSD device, did not consider using the cache operation to the relatively high frequency accessed objects.

MDS cluster load balancing application mainly have NFS [12] as the representative of the directory subtree partition, and zFS [13], GFS [14] as the representative of the Hash distribution. Both of them make good use of the locality principle, able to traverse the file quickly. But the load distribution of the former requires the administrator to specify, cannot be timely adjustments when the load changes. The latter design is more complex, data synchronization between nodes is usually introduced an additional overhead in order to ensure data consistency, therefore, in some small and medium storage system performance is not efficient enough. Hash distribution using a files absolute path or other unique identifier to determine the metadata storage location. The advantage of this strategy is simple, efficient search and the client can directly locate the file, keeping the average case load balancing. Because it is hashed file distribution completely, effectively avoid the problem of centralized access to a single directory caused by localized access. However, large amounts of data migrate when the number of servers changes.

To solve the above problems, this paper proposes a storage cache implementation, based on the Object-based distributed file storage system, we design an object-based storage caching (OBSC), assign the hot files to the better performance storage devices. On the basis of OBSC, we consider the problem of the hot data object's evaluation, analyzed and presented a hotspot replacement policies based on the frequency of access, improve overall system file IO rate. For the problem of cluster load imbalance caused by performance heterogeneous, proposed a metadata server hotspot evaluation model, mainly to solve the problem of getting service load conditions exactly. We also proposed a strategy based on weight Load balancing, mainly to solve the problem of load balancing in case of server performance heterogeneous.

Section 2 description the design and implementation of OB SC system; MDS cluster load balancing strategy proposed in section 3; Section 4 present the experiments and analyses the results; finally we concludes the paper work .

2. OBSC Caching Method

2.1. The Overall Architecture of OBSC

In view of the hot file caching issues under the heterogeneous storage cluster, based on object storage system, combined with network caching technology. This paper presents a method for hot files cached, the purpose is to improve the throughput and load capacity of the system. The system architecture model as shown in Figure 1,the entire system based on object-based storage system architecture constitute of the client , MDS cluster and the OSD cluster. In the OSD cluster, the better performance of the OSD device deployed as a proxy server (Cache proxy), allowed access to the high-frequency data object cache.

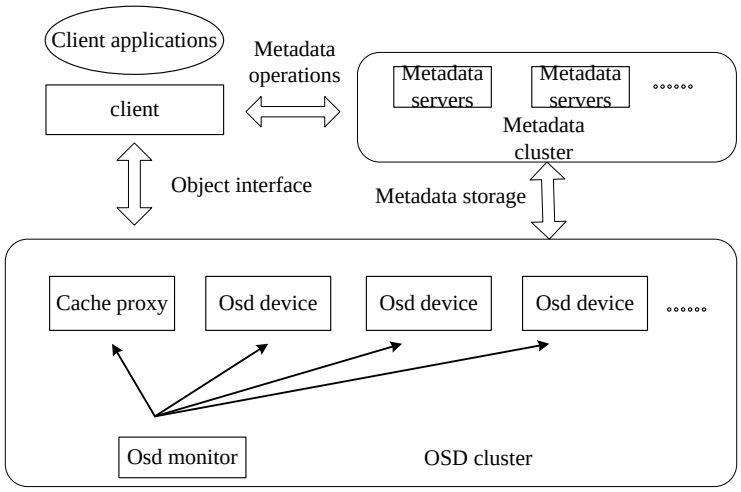

Figure 1. The Overall Architecture Of OBSC

Clients of the system must install the client software, so the system can show a single-level file storage space to users, make the application on the client can use standard file operation interfaces. Metadata management platform is installed on a metadata server cluster, metadata stored in the OSD be managed and responsible for coordinating the name space of entire file system. OSDs in the storage cluster are installed object-based file system, such as btrfs[15]. These file systems manage logic block distribution of local data objects, to provide object-based file-interface. Storage cluster proxy servers and ordinary OSD devices have the same software configuration, provide the same object-based management capabilities. The OSD monitors maintain the equipment and management information table and authorize the proxy server by maintaining the deployment of the entire storage cluster scheduling mechanism. Device information table records the information stored for each device, including storage capacity, grouping, i-node list of stored data objects, directory information and the device's transmission delay, etc. Then the device information table according to its transmission delay and backup group to decide which units OSD device deployed as a proxy server. Device information table needs to be updated in accordance with a certain cycle. To update the device information table by timestamp mechanism, and follow the latest timestamp device information table storage cluster scheduling.

Between the client and metadata server are generally about the data objects requested metadata information queries, and the file system namespace changes, etc. Storage cluster provide to the client object-based interface, clients can use these interfaces directly read data objects. Metadata server is responsible for managing the storage cluster storage.

In OBSC caching system, when the client object reads the data from the storage cluster, first find out whether a cache hit, if hit it directly to obtain higher transmission bandwidth through a proxy server, thereby reducing network latency overhead, if cache misses in accordance with the prototype system data access methods to read data, as shown in Figure 2. The general configuration of proxy server and OSD device is the same. After applying the method, OSD storage capacity and operating mechanism does not significantly change, but its transmission efficiency has been improved because of the proxy server to cache hot objects. Without altering the mechanism on the basis of its original, maintaining the reliability and scalability of the storage system, to maintain the functional properties of original system, also improve the file I/O performance of the system.

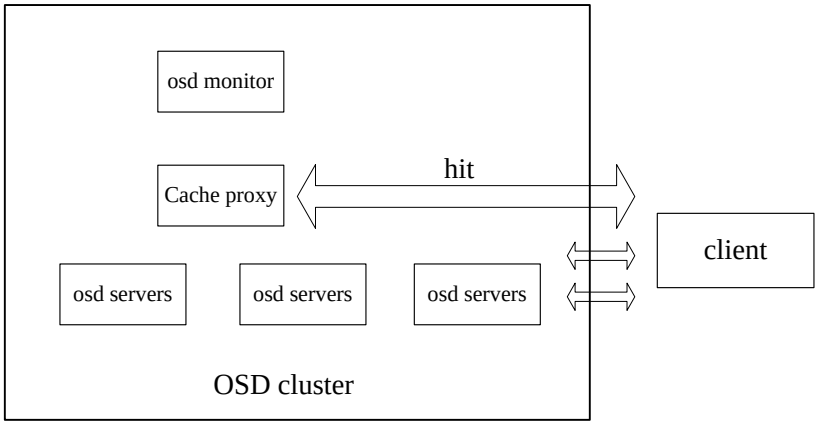

Figure 2. Data Access Method For Storing Cluster

2.2. Replacement Strategy based on Access Frequency of Hot Spots in OBSC

Because traditional cache data replacement strategy is not ideal here, this paper presents a data object replacement strategy based on popularity, compared with the conventional LRU [16-19] and LFU [20-21] replacement strategy, the policy considering data object access frequency, interval and object size, try to cache the recent popularity relatively high data object, set popularity as H, M is the object request into the cache, s[n] represents the size of the object, specific strategies are described below:

(1) In the storage cluster monitor, the data objects in cache sort by popularity H in descending order. Proceed to step 2.

(2) Suppose there are n data cache objects, consider whether you need to replacement data object M. If Hn > Hm indicate that popularity M is relatively low, cannot be replaced. If Hn < Hm, proceed to step 3.

(3) Find a maximum A in the sequence of cached data objects, so that the sum of A to N objects' space usage larger than M's space usage, if Ha > Hm, shows the object M's space is too large, the objects of the same capacity within cache popularity is higher than M, so cannot be replaced. If Ha< Hm ,indicate the object M's popularity is larger than the objects of the same capacity within cache, so the object A to N swapped out, swapped into M.

The pseudo-code description of the policy is shown as Figure 3:

```
The cache data objects in descending order according to the popularity H, get the
number of columns in descending order H[]
Judge whether to replace the data object M
If（H[n]>Hm） then
No replacement, end; /* Popularity M lower than H[] sequence's minimum
                        value H[n], not replaced */
else
S←s[n]        /* Find a maximum A so that the sum of A to N objects' space
           usage larger than M's space usage */
i←n
while（S>sm）
do i←i-1
S←S+s[i]
a←i
if(H[a] >Hm) then
No replacement, end; /* Object M's popularity is lower than the
     objects of the same capacity within cache, do                    not
```



```
swapped */
    else Object M replace object A to N, enter the cache.
```

Figure 3. Popularity Replacement Algorithm Based On Frequency Of Access

In the overall architecture of OBSC, the strategy only involves storage cluster monitor, all calculations and deployment scheduling process carried out in the Monitor.

3. MDS Cluster Load Balancing Strategy

3.1. MDS Performance Evaluation Model

Traditional performance model write server performance into the configuration file as a static value, you cannot dynamically adjust it during system operation. To address this problem, this paper proposes a real-time performance model based on operation delay. Operating delay is an important indicator of QoS[22], which is defined as: the combine of transmission time of data packets between server and client and the processing time of data packets on the service time. Operating delay may be influenced by network conditions, the impact of server hardware and software configuration and other factors.

MDS of stronger performance process meta data operations in shorter time, then service request of same queue length can be completed in a shorter time, and the reflection is a smaller slope of the mapping curve between parallel access requests and the average operating time delay requests (denoted as <pIO , latency>). When the system allocates its workload, MDS of minimum <pIO, latency> slope curve guarantees the minimum delay increase, and therefore the MDS of smaller slope curve should have a higher weight value in the allocation of the workload. [23] shows that with an increase in the number of concurrent access, the average delay of the operation of the server subsequently linearly increased. This is due to under the assumption of stable network condition and processing capability, the time delay of metadata service request increase linearly with the increase of waiting queue length.

Based on above, this paper use <pIO, latency> slope curves to indicate MDS metadata processing capability. When the system is initialized, each MDS heat is initialized to be 0; During system running, MDS record parallel requests sent to it and the average time of operations delay at this time; due to system performance jitters, testing error or other reasons, the collection of discrete sample values, though not presented as accurate straight line, but show the trend of linear regression overall. Therefore, we use the least squares method[24] to collected data on the linear fit, and use the slope of the curve was fitted as the main parameters of the performance model.

Slope least squares estimated value is calculated by the following:

First, we assume that the request latency (Y) and the parallel request (X) relationship are expressed simply as:

$$Y = a + bx + \varepsilon \tag{1}$$

The above equation (1) represents a random error vector by ε and assume as $\varepsilon \sim N(0, \sigma^2)$, Vectors X and Y respectively expressed as $X = [x_1, x_2 \cdots x_n]^T$ and $Y = [y_1, y_2 \cdots y_n]^T$.

$$Slope = \hat{b} = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} = \frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)\left(y_i - \bar{y}\right)}{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2} \tag{2}$$

To prove this theory, this paper carried out two experiments on <pIO, latency> mappings under different network bandwidths and different CPU frequencies. MDS

performance may be affected by network bandwidth and CPU frequency, this paper hopes to observe these server resources to verify <pIO, latency> curve affecting factors.

Figure 4 is three <pIO, latency> curves under different network bandwidth. The experimental results show that with the network bandwidth increased from 50Mbit/s to 200Mbit/s, <pIO, latency> slope of the fitted curve monotone decreasing; Figure 5 is two <pIO, latency> curves under two different CPU frequencies, which also show that <pIO, latency> fitting curve slope and MDS CPU frequency present a negative correlation. The two sets of experiments show that with increase of the allocation of resources, MDS performance enhanced linearly, corresponding <pIO, latency> fitting curve slope decreases continuously. According to experimental results, MDS delay performance model based on the metadata request can accurately reflect MDS metadata processing capabilities.

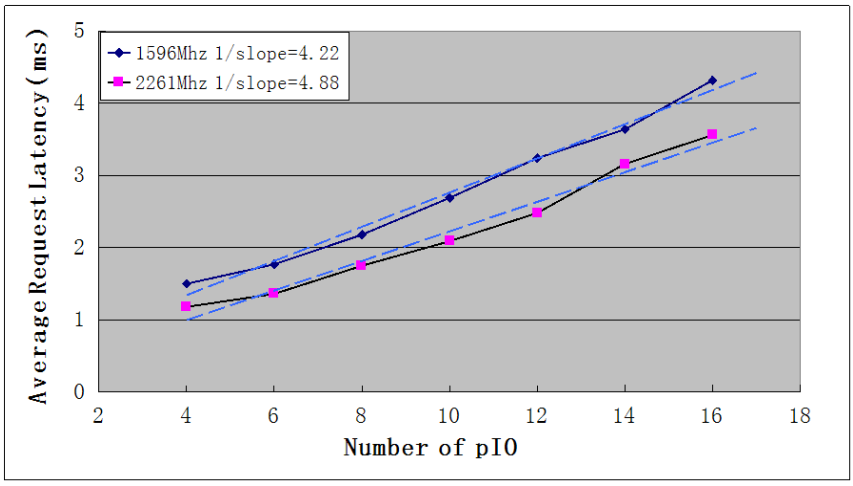

Figure 4. Under Different Network Bandwidth Measured MDS Metadata Average Operating Time Delay

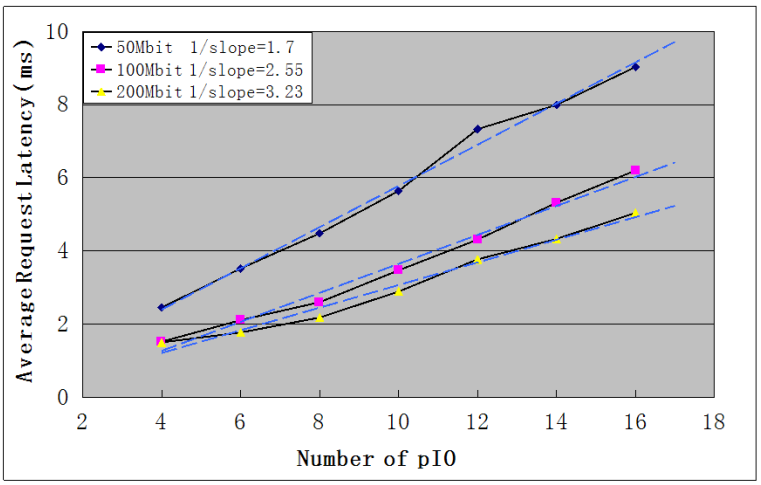

Figure 5. Different CPU Frequency Measured MDS Metadata Average Operating Time Delay

3.2. Weight-Based Metadata Server Cluster Load Balancing Strategy

Current MDS cluster load balancing strategy lacked of research on MDS cluster performance differences. For such a situation, this paper presents a weight-based MDS cluster load balancing policy. WLBS drawn server performance and value by

performance model described earlier, and use it as weight to allocate the overall system load. During MDS cluster running, when the load changes, WLBS dynamically move overloaded nodes to idle nodes by directory sub-tree migration method. In order to achieve the goal of load corresponding performance, MDS higher than the average load level automatically proceed load migration, move part of the directory sub-trees to MDS with load below average load level. This method avoid weak cluster node overload and being bottleneck of system performance and stronger nodes not fully utilized, thereby maximizing system metadata processing capability and achieve MDS cluster load balancing eventually .

Each MDS has a daemon to count service requests handled by MDS, recording the time of each request. Each heartbeat cycle time out, the metadata server broadcasts its current load levels (Work_load) and weight (W) to all other cluster metadata server via UDP protocol, and record sending time. Heartbeat cycle parameter settings can be configured by the metadata server based on the actual situation. After completion of receiving broadcast information from other servers in the cluster, metadata server calculate the target load. It is calculated as follows:

$$target\_load_i = \frac{W_i}{W_{total}} * total\_load \tag{3}$$

In the above equation, target_loadi indicates target load of server no. i, the performance ratio of the performance values. Wi is weight of server no. i, Wtotal is summation weight of all the metadata server cluster, total_load is summation of all metadata servers current load in the cluster.

The following function is used to measure the degree of MDSi unbalanced load:

$$IM_i = | my\_load_i - target\_load_i | (i \in M(n)) \tag{4}$$

In the above equation the set M(n) represents MDS cluster, the cluster number of ||M(n)|| = N. IMi indicates the degree of MDSi load imbalance and is the absolute value of MDSi current load and target load differences. IMi> 0 indicates that the node overloaded and requiring exporting directory subtree, Mi <0 indicates that the node can receive more loads. IMthreshold indicates the degree of imbalance. For any MDSi in the MDS cluster, when the current load of the server in step 5 of the load subtracted from the target, if the difference is positive and larger than IMthreshold, and lasts for two cycles or more, the load balancing processes will be started. This is due to the load migration itself will bring some of workload, these rules can improve the trigger conditions of load migration. This avoids too frequent migration between MDSs and unnecessary overhead and system performance jitter.

After determining load imbalance degrees of each MDS, WLBS adjust MDS load distribution and determine the target MDS object of sub-tree migration by directory subtree migration. The main process is as follows:

(1) Initialization

1 Importer_set= Φ;

2 Exporter_set = Φ;

3 for i←1 to N

4 target_loadi←(Wi / Wtotal)*total_load / * target load with weights proportional MDS * /

5 if my_loadi - target_loadi ≥ IMthreshold;

6 Exporter_set←Exporter_set ∪ MDSi; / * put the node into the output load overload set * /

7 else if target_loadi - my_loadi ≥ IMthreshold

```
8    Importer_set = Importer_set ∪ MDSi;   / * the load lighter nodes in the input
set * /
```

(2) Determine the migration between node mapping

```
1 for each i ∈Exporter_set   / * set of nodes in ascending order according to IM
absolute value * /
2    if IMi ≤ IMthreshold
3       continue;
4    for each j ∈ Importer_set   / * set of nodes in ascending order according to IM
absolute value * /
5       if IMj ≤ IMthreshold
6          continue;
7       TRY_MATCH(MDSi,   MDSj);
8       if IMi ≤ IMthreshold
9          break;
```

(3) Migration between heat matching nodes

```
1 MATCH(MDSi, MDSj)
2    Pmig = MIN(IMi, IMj)
3    IMi = IMi - Pmig
4    IMj= IMj -Pmig
5    MIGRATE(MDSi, MDSj, Pmig)   / * put the load to the directory subtree migrate
migrate from MDSi to MDSj * /
6    return Pmig
```

Sets Export_set and Import_set store overload and light load nodes respectively, where each overloaded node MDSi has an overdrive value IMi, needing searching for introduce matching overload heat target MDS object in Import_set node. Due to overload output load node and light load not correspond to the introduction of the amount of nodes, in most cases it is a "one to many", the load migration process need multiple nodes coordinating to be completed. For example, MDSi need to output a large load, more than any light-load node can be loaded, then overload heat migrated to multiple target nodes respectively. Migration process will first search the directory sub-tree of Pmig sub-tree sum load in MDSi, and then transferred it to the light load node MDSj. Each directory subtree represents a certain amount of load, while after the migration customers want to access the transferred directory subtree through MDSj, therefore the directory subtree migration means the transfer of workloads. So that each MDS node be assigned corresponding proportion of work load according to its own handle ability, then MDS cluster reach balance eventually.

4. Future Work

In this paper, heterogeneous storage environments put forward new ideas, designed and implemented a new cache method, proposed WLBS Policy used in heterogeneous MDS environments, its effectiveness and feasibility has been verified by benchmarking under the experimental environment. But throughout the course of the study there are still some issues that need further study and discussion, as presented below following the work of the future:

(1) On the basis of OBSC herein studies, can be further used with more reliable mathematical model simulate and predict the user's high frequency access files.

(2) Throughout cloud storage system architecture, we studied the storage cache of object-based distributed file system belonging to the file system layer. In the future, the underlying block-level cloud storage system combinations to build cloud storage

platform, also can use a standard interface provided by the client file in the file system according to the needs of the development of the upper user applications.

(3) For heat threshold WLBS involved, heartbeat cycle, Slop update cycle configuration parameters still needs to be optimized according to the results of the experiments.

Acknowledgments

This paper is supported by Natural Science Fund of China under grant No.61202094 Zhejiang Provincial Natural Science Foundation under grant No. LY13F02004, No.LY16F020018, China Postdoctoral Science Foundation Funded Project under grant No. 2013M541780. Key Projects in the National Science & Technology Pillar Program (No. 2014BAK14B00).

References

[1] R.O. Weber, "Information technology-SCSI object—based storage device commands (OSD)",Technical Council Proposal Document T10 / 1355-D,Technical Committee T10, Aug 2002.
[2] M. Mesnier, G.R. Ganger and E.R. Jedet, "Object—based storage, IEEE Communications Magazine",V01.4 1:PP.84—90,2003.
[3] F. Wang, S. Brandt, E. Miller and D. Long, "OBFS:a file system for object-based storage devices. In Proceeding of 1 2th NASA Goddard", 2 1 st IEEE Conference on Mass Storage Systems and Technologies, April 2004.
[4] S. Quinlan and S. Dorward, "Venti: A New Approach to Archival Storage", In Proceedings of the FAST 2002 Conference on File and Storage Technologies **(2002)**.
[5] G. Barish and K. Obraczka, "World wide web caching: trends and technologies, IEEE Commun", Mag.Internet Technol. Ser. **(2000)** 178–185.
[6] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms", in: Proceedings of 1997 USENIX Symp. Internet Technology and Systems, 1997.
[7] L. Fan, P. Cao, J. Almeida and A.Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol", IEEE/ACM Trans. Netw. 8 (3) **(2000)** 281–293.
[8] D. Nagle, D. Serenyi and A. Matthews, "The Panasas ActiveScale storage cluster-delivering scalable high bandwidth storage", In Proceedings of the 2004 ACM/IEEE Conference on Supercomputing(SC `04), November 2004.
[9] "Sun Microsystems", Lustre. http://wiki.lustre.org/index.php?title=Main Page.
[10] P. Schwan, "Lustre: Building a file system for 1000-node clusters", In Proceedings of the 2003 Linux Symposium, July 2003.
[11] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long and C. Maltzahn, "Ceph: A scalable,high-performance distributed file system", In Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI), Seattle, WA, Nov. 2006.USENIX.
[12] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel and D. Hitz, "NFS Version 3 Design and Implementation[C]", In Proceedings of the Summer USENIX Conference, 1994: 137-152.
[13] O. Rodeh and A. Teperman., "zFS - A Scalable Distributed File System Using Object Disks[C]", Mass Storage Systems and Technologies, 2003:207-218.
[14] S. Ghemawat, H. Gobioff and S.-T. Leung, "The Google file system[C]", In Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP' 03), 2003:29-43.
[15] Btrfs. http://en.wikipedia.org/wiki/Btrfs
[16] E. O'Neil, P. O'Neil and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering", in Proc. of the ACM SIGMOD International conference on Management of data, Washington, D.C., May 1993.
[17] D. Lee, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho and C. Kim, "LRFU:A spectrum of policies that subsumes the least recently used and least frequently used polices", IEEE Transactions on Computers, vol. 50,no. 12, pp. 1352–1361, 2001.
[18] N. Megiddo and D. Modha, "ARC: A self-tuning, low overhead replacement cache", in Proc. of the 2nd USENIX Conference on File and Storage Technologies, San Franciso, CA, March 2003.
[19] B. Gill and D. Modha, "SARC: Sequential prefetching in adaptive replacement cache", in Proc. of the 2005 USENIX Annual Technical Conference, Anaheim, CA, April 2005.
[20] S. Jiang, X. Ding, F. Chen, E. Tan and X. Zhang, "DULO: An effective buffer cache management scheme to exploit both temporal and spatial locality", in Proc. of the 4th USENIX Conference on File and Storage Technologies, San Franciso, CA, December 2005.

[21] X. Gu, B. Veeravalli and W.J. Lin, "Design and analysis of practically realizable dynamic data allocation and replication algorithm with buffer constraints for distributed databases", IEEE Trans. Parallel Distrib. Syst. 17 (9) (2006) 1–1
[22] D.A. Menascé, "QoS issues in Web services", IEEE INTERNET COMPUTING[J], 2002, 6(6):72-75.
[23] A. Gulati, C. Kumar, I. Ahmad and K. Kumar, "BASIL: Automated IO Load Balancing Across Storage Devices[C]", FAST'10 Proceedings of the 8th USENIX conference on File and storage technologies.
[24] P. Zhu, Q. Zhong, W. Xiong and B. Xu, "A Robust Least Squares for Information Estimation Fusion", JDCTA: International Journal of Digital Content Technology and its Applications, Vol. 5, No. 4, pp. 65-73, 2011.

Authors

Jilin Zhang, he received the PhD degree in Computer Application Technology from University of Science Technology Beijing, Beijing, China, in 2009. He is a PostDoc at Zhejiang University. He serves as an assistant professor of software engineering in Hangzhou Dianzi University, China. His research interests include High Performance Computing and Cloud Computing.

Yucheng Su, he is now M.S. in School of Computer Science and Technology in Hangzhou Dianzi University, China. His research interests include High Performance Computing and Cloud Computing.

Jian Wan, he received the PhD degree in Computer Application Technology from Zhejiang University, Zhejiang, China, in 1989. He is currently a professor in software engineering in Hangzhou Dianzi University, China. His research interests include Grid Computing, Service Computing and Cloud Computing.

Li Zhou, she serves as an assistant professor of software engineering in Hangzhou Dianzi University, China. His research interests include High Performance Computing and Cloud Storage.

Jue Wang, he is currently working as associate research fellow in the supercomputing center of Chinese Academy of Science. The motivation behind his work is to improve soft systems by increasing the productivity of programmers and by increasing software performance on modern architectures including many cores clusters and GPU.

Lei Zhang, she received the Ph.D. degree in School of Information and Communication Engineering from Beijing University of Posts and Telecommunications, in 2009.She is currently lecturer in the Computer Science Department, Beijing University of Civil Engineering and Architecture. Her areas of interest are in wireless sensor networks, data security, routing and multicasting protocols, and data mining. She is also a Member of China Computer Federation (CCF).