International Journal of Grid Distribution Computing
Vol.7, no.3 (2014), pp.89-102
http://dx.doi.org/10.14257/ijgdc.2014.7.3.09

Design and Implementation of Metadata Cache Management Strategy for the Distributed File System

Jingning Liu[1], Junjian Chen[2]*, Wei Tong[1], Lei Tian[1] and Cong Chen[1]

[1]School of Computer Science & Technology, Huazhong University of Science & Technology,Wuhan Natl Lab Optoelect, Wuhan, China
[2]School of Electronic & Information Engineering, Foshan University, Foshan, China
j.n.liu@163.com, cjj_hust@163.com, weitong@163.com, ltian@hust.edu.cn, chenconghill@163.com

Abstract

Caching strategies and the resulting cache coherence control technique have become the key techniques for the system development. Based on the study of the cache technology of the home and abroad distributed file systems, and combined with the design requirements and characteristics for our multi-user parallel file system Cappella, this paper proposes a new meta-data cache management strategy. And this strategy not only improves the performance of metadata services, reducing the user access latency, but also controls metadata cache consistency very well.

Keywords: *Distributed File System, Metadata Cache Management Strategy*

1. Introduction

In such an era of information explosion, it is especially urgent and important that providing mass data storage service with high performance, high reliability, high expansibility and high security. Researched and developed by Huazhong University of Science and Technology Wuhan National Laboratory for Optoelectronics independently, Cappella, a distributed file system based on object, is a mass data storage system aimed at supporting high concurrent access, high aggregate bandwidth and high availability.

Cappella distributed file system provide POSIX file interface to realizing remote access and management of data. Clients, MDSs (MDS) and object-based devices constitute the overall system. Clients run below the VFS as loadable modules. Users' access requests Transfer to the Client file system through VFS, and achieve specific file access. Before reading the file data, the file system must traverse from root directory, and obtain directory entry of data in each ancestor directory in proper order. After found files describe information and space distribution information, data access is finally implemented. The information except file data is metadata, used to describe the file data. Before accessing data, file system must visit the corresponding metadata. In order to reduce server load and fully utilize the network bandwidth, access optimizations have processed separately according to data and metadata differing access properties. Cappella makes data and metadata transmission path separated, and use special MDS to realize the metadata storage and management. After

* Corresponding author.

ISSN: 2005-4262 IJGDC

obtained the information such as the space layout of the file, data transmission is in progress between Client and equipment directly. Figure 1 shows the framework of Cappella.

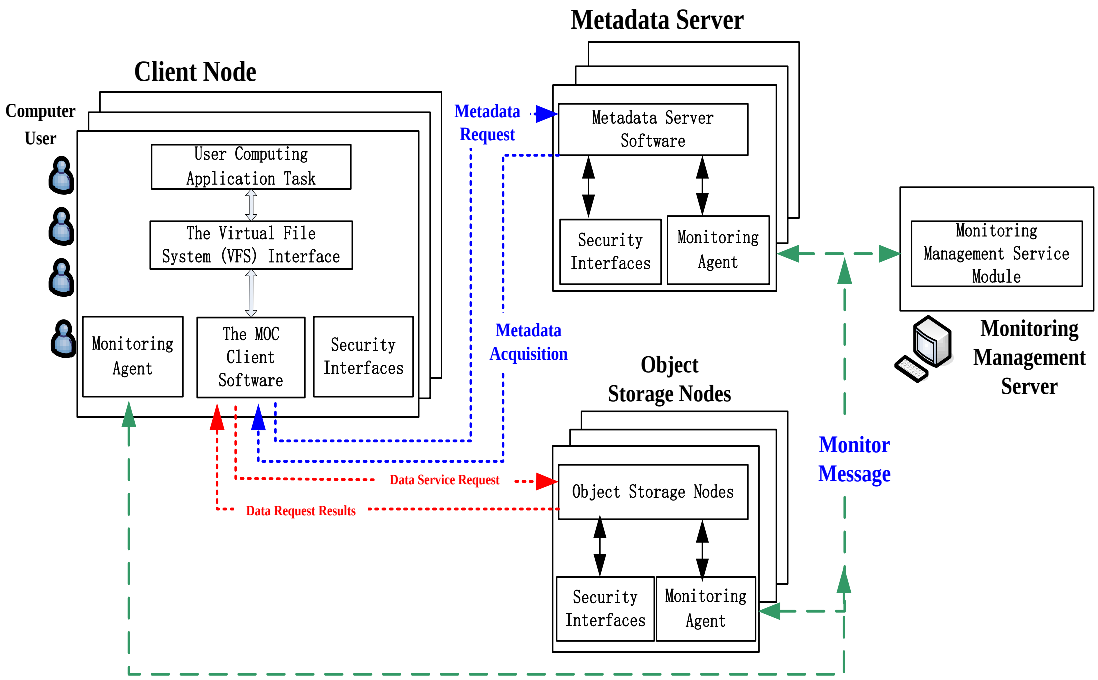

Figure 1. Framework of Cappella

Different from traditional distributed file system, Cappella provides object-based access interfaces, for taking full advantage of object to realize the data high-performance parallel transmission and storage. Details of object can be found in Literature [1, 2].

MDS, which is indispensable for file access, is serving an important function of the entire file system. Its main functions are: First, effective organization and management metadata, providing the correct data access services. Corresponding to every Client file access request, such as mkdir and unlink, MDS all needs to properly modified or writing corresponding files or directories metadata and return the final results to the Client. When some nodes fault, it needs other nodes timely replace fault machines continue to provide metadata services, guarantying system availability. Next, maintain globally unique name space. As a multi-user file system, Cappella needs to ensure all files and directories have unique names for the user access, and all users seen filename and properties are the same. Cappella use MDS cluster unified provides services. The file system name space is divided into different set, managed by different MDS nodes separately. It needs ensure consistency between each node metadata and system metadata load balance. In addition, achieve global equipment information management and scheduling. MDS cluster of Cappella should timely detect dynamic join or exit of equipments, and make the latest device list information feedback to the Client, to ensure the Client normal access services. Meanwhile, it needs to realize the management and operation on each node of MDS cluster. Final, MDS should also realize safely control of user access.

According to the research done by University of California, Berkeley and Carnegie Mellon University researches, the Auspex server running NFS v2 and AFS's SPARC workstations running specialized load research [3-5], the statistics show that, in the file system access, the meta-data operations respectively of NFS and AFS operation account for 75.4% and 82.2% of the total requests, and as to occupancy rate of the system resources is respectively 49%, 63.5%. In addition, we can see that in document [6], the research about the IO load of distributed parallel file system Hamel and Tachyon show that among different kind of files , in home storage, the catalog file possess 8.48% in 2006 and increase to 15.26% in 2009, in scratch storage, the percentage grow by 1.51% over the same period. We know that in the file system the number of requests of metadata is far bigger than the amount of data access requests. Besides, with the change of system application environment and scale, the proportion would increase and has a growing trend. Therefore, how to ensure efficient service of MDS has become a key design point in distributed file system.

Caching mechanism has been widely used in the distributed file system as an important means to speed up file access speed.

Take a look at the ls command supported by the file system to analyze the process of system calls in the Client file system. Assume the existence of the directory /d1, and the existence of subdirectory /d1/d2, /d1/d3 and /d1/d4, executing ls command on /d1, first, the Client will perform lookup operation on /d1 to get /d1 metadata. Then command "readdir /d1" will be sent to the MDS to get the catalog data of /d1. After that, according to the catalog data observed, lookup operation will be executed respectively on the subdirectory. Figure 2 shows the specific process:

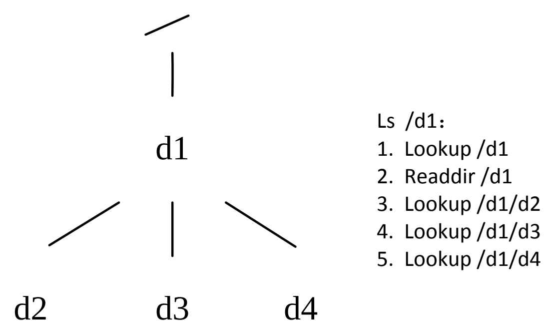

Figure 2. Readdir Examples

In the example, if the Client does not support metadata cache, the lookup operation will be sent to the MDS and returns the corresponding metadata to the Client every time. As to a look up operation, there would be 5 metadata requests through the internet. But if the Client support the metadata cache and each lookup operation hits, there would only be 1 metadata request through the net. For large catalog, the Client metadata cache technology will have more prominent advantages.

Metadata cache technology brings performance improvement without doubt, however, it brings a problem—cache consistency control. In Multi-user file system, users may get access to the same file or catalog, leading to multiple copies of the same metadata in different Clients. In this situation, if we do not control when one Client update the metadata, metadata cache consistency will be destroyed, this would result in the error of system service.

The purpose of this article is to study and implement a new kind of distributed file system metadata cache management strategy, to provide efficient metadata service and to maintain multiple Clients' metadata cache consistency. Test results show that, after adopting cache management strategy, performance of cappella file system metadata service promotes 2 to 3 times. And in the same test environment, the metadata supported by IOPS is obviously better than Luster.

2. Related Work

Cache management policy plays a very important role for a distributed file system performance and reliability. The cache can effectively reduce the network traffic overhead and reduce the cost of computer resources of MDS, as well as improve the capacity of MDS cluster to provide services outside. Existing distributed file systems also have their own different cache consistency control schemes. The following is analysis on cache strategy used by Lustre [7], NFS [8, 9] and Sprite [10].

NFS is a stateless distributed file system, and server handles all requests for data and metadata. In Its stateless server model, the server does not retain information about customers, and all file data is written to disk synchronization. Because the server does not record which customers are using the file, it cannot guarantee the cache consistency. An NFS Client periodically detects whether a file has been modified, if the file has been modified, the Client will set the file cache is invalid. Because the NFS Client cannot confirm whether there are other customers accessing the same file, all consistency checks are performed by the file server. Therefore, whenever a Client modifies a file, it must immediately write back to the server - as known as "write through" policy.

NFS v4 uses entrusted mechanism to manage Client cache, Entrusted is a technique used by server entrusting file management to Client. Server can award the Client read entrusted or written entrusted. Read entrusted can been awarded to multiple Clients, and they are compatible to others. Writing entrusted could only be granted to a Client, and they are exclusive to other writing entrusted and read entrusted. Because reading entrust guarantees that there won't have a Client modify current visit the documents, the Clients who have obtained read entrusted can read directly from local caching data and metadata, without having to launch a request to the server. Write entrusted guarantee that only one Client has access to the file, so the Client gets the writing entrusted can directly update local data and metadata, the Client will refresh information and metadata to the server when it waiver or the server the server recalled to entrusted. The Entrusted mechanism significantly reduces interaction for commissioned file between the server and the Client.

As the improvement of NFS, Sprite uses the state server model. Different from NFS, Sprite support explicit open and close operation. Through tracking open and close operation, Sprite file servers know all customers for the use of the files, and know whose potential to write. According to records of the file sharing access, Sprite separately uses different cache strategy: For not sharing write documents, it allows the Client cache and doesn't require execution write-through. If the file sharing write, sprite will not allow Client cache it, it must execute files and metadata write-through.

Lustre is a distributed file system based on object. It uses Lustre Distributed Lock Manager (LDLM) to complete the synchronization and consistency control of sharing resources access. In view of the different share access situation, Lustre uses two different locks model for management of Client cache. In order to improve the treatment efficiency of metadata, For catalog do not be often shared access, Lustre awards the Client Sub Tree Lock(STL), and achieve Client metadata write back to cache strategy. The Client who has directory's STL can cache metadata, and local access and update the metadata, until appears lock request conflict or Client cache inadequate, then brush the latest metadata back to MDS. For frequent shared directory, lock server will grantee intent lock. When the Client sends requests for metadata operations to MDS, Indicating operating intention, MDS executes operation according to request intention, and the Client will no longer cache metadata.

NFS Client uses write wear strategy to synchronize the update of all data and metadata, and ensure the consistency of the documents state strictly. This method is easy to implement and control, because the server does not need to maintain any access information. To a certain extent, the method reduces the server's burden. However writing update must penetrate network to synchronize the server every time, which will result in a huge network transmission and disk IO, consuming time. On the basis of NFSv4, NFS introduced entrusted mechanism and significantly reduced the entrusted files interaction between the Client and the server. Sprite as the improvement of NFS, use a state server model and grant different cache permission by supporting open and close operation to track the Client's file operation. Compared with the file access, NFS Sprite gets a boost performance. But Sprit is just a study and gets no fusion to NFS ultimately. Lustre uses distributed lock manager to grant Client corresponding cache privileges, according to different types of sharing access situation. It is flexible and efficient. But because of the distributed lock's complex realization, in order to reach a consistency allocation of the lock resource, the operation would bring great resource expense.

3. Cappella Metadata Cache Management Model

3.1. Problem Analysis

In order to speed up the operations of metadata and provide IOPS as high as possible, Cappela uses secondary cache structure of metadata, as shown in Figure 2. First, it establishes local metadata cache based on the underlying metadata stored on the server. Then, it allows Client caches to obtain metadata information from MDS.

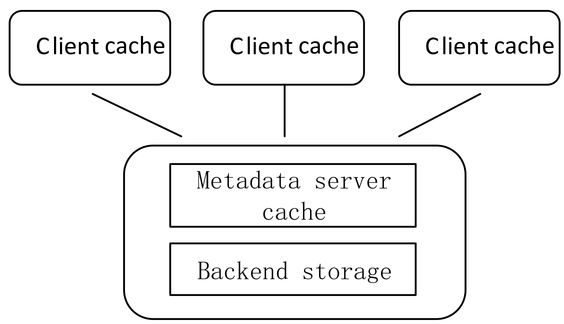

Figure 3. Cappella Metadata Cache Structures

As shown in Figure 3, when the Client issues the metadata request, it will first check its local cache. If it hits the target, the Client will directly read the corresponding metadata and execute subsequent operations, otherwise, Client issues the metadata request to MDS, when the MDS receives the request, it will check the upper cache, if it hits the target, it sends metadata directly back to the Client, otherwise, it reads the metadata from the backing storage. Then, it inserts the metadata to the upper cache and sends the metadata to Client. If in most cases, the Client can hit the cache, it will reduce the number of getting the metadata through the network, as well as reduce the load of the MDS.

By analyzing the secondary metadata cache model of Cappella, we know that metadata cache consistency issues include the following two aspects:

(1) The consistency between upper cache and the underlying storage in MDS.

(2) The consistency between Client cache and MDS.

For each metadata operation from the Client, like mkdir, create, MDS package them as a series of atomic operations. The atomic is ensured by the locks of concurrency control module in the MDS. For each operation, MDS will check the upper cache, if it hits the target, then it locks, for operation of reading metadata, it sends metadata directly back to the Client, and it unlocks before sending metadata back to the Client. For operation of writing metadata, it updates the underlying database first, then it updates the upper cache, before sending the updated metadata, it unlocks. If it misses the target, it reads metadata from underlying database, and establishes structure of upper cache. The subsequent operations are the same as above. Atomic operation ensures accuracy when a number of Clients simultaneously access the metadata. At the same time, the MDS's log module will record information about every step of the metadata operation, If the system failures when execute operation of metadata, Log module will use the log information to restore the system to the consistency state, o ensure the correct of upper cache and underlying storage.

The cache consistency between Multi-Clients and MDS becomes the key and difficult issues. Cappella, as a multi-user parallel file system, in some case, there may be different users to simultaneously access the same file or directory , in order to improve access efficiency, the Client will cache the metadata obtained from MDS, and may update local cache while not notifying the MDS, then it will destroy the consistency of multi-Client cache. Customers are transparent with each other, and they do not know about each other. When a customer updates local metadata, other customers will be out of date cached information. The main purpose of this paper is to address metadata cache consistency of multi-Client.

3.2. The Design Scheme

Client sends requests for metadata to MDS, then MDS executes a battery of operations, then returns the corresponding metadata to Client and authorizes the Client the different cache permissions of the metadata cached. Here we define four kinds of cache permissions as below:

Table 1. Metadata Cache Permissions

Name	Meaning	Value
NO_CACHE	Client can’t cache the metadata item	0x0000
READ_CACHE	Client can cache and only read the metadata item locally	0x0001
WRITE_CACHE	Client can cache and modify the metadata item locally	0x0002
CREATE_CACHE	Client can create new directories and files under the directory locally	0x0004

Permission CREATE_CACHE includes permission WRITE_CACHE, READ_CACHE, permission WRITE_CACHE also includes permission READ_CACHE. Figure 4 shows the relations among all the permissions.

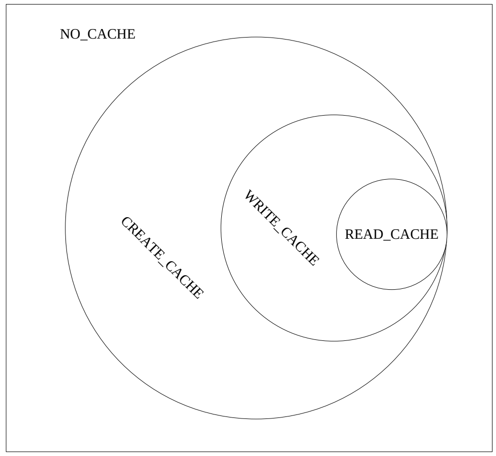

Figure 4. Relations Among Metadata Cache Permissions

The default permission is NO_CACHE, which means that Client can't cache the metadata. In this mode, every metadata item Client gets from MDS is only effective when first used, and then the metadata item is discarded. If the same metadata item is accessed again, the Client must send a new request for it to MDS. Clients caching the same metadata item all can be authorized the READ_CACHE access permission, under this condition, when one Client needs to modify the metadata item, it should send a request to MDS. MDS has cached information of all Clients who have cached the metadata item, so MDS will send messages to all Clients cached this metadata item that the item they cached locally is invalid. Receiving the responses of all Clients, MDS begins executing the request send by the Client who wants to modify the metadata item. Permission WRITE_CACHE includes permission READ_CACHE, while means that the Client who can modify the metadata item locally can also read the metadata item locally. When conflicts of accessing occur, MDS withdraws the Clients WRITE_CACHE access permission and asks it to commit the newest metadata item to MDS, then MDS authorizes READ_CACHE permission to other Clients.

CREATE_CACHE is the highest authority that Client can get, authorizing Client to create files and directories under the cached directory locally. It applies to the situation that the directory is allowed to be accessed by only one Client, and the LOCAL_ID space should be pre-allocated (Cappella file system uses a global id—— <MDS_ID,COLLECTION_ID,LOCAL_ID> which contains three fields to identify a unique file or directory. The file system uses MDS group to provide metadata service, the namespace of whole file system is divided into some mutually disjoint subtree managed by different MDS, and each MDS may manage one or more subtree. The field MDS_ID of global id identifies which MDS in the group the metadata belongs to, the subtrees in the same MDS are identified by the COLLECTION_ID, and LOCAL_ID identifies the file or directory under each subtree).

Permissions WRITE_CACHE and CREATE_CACHE allow Clients to update the metadata items they cached locally (CREATE_CACHE also allows Clients to create new files and directories locally), then commit the new metadata items back to MDS at a certain time, that will decrease the overhead of metadata updating by large. Permission CREATE_CACHE will increase the speed of creating files and creating directories.

Permission READ_CACHE applies to files and directories, indicating that Clients can read the metadata item locally, WRITE_CACHE applies to updating file's metadata items, CREATE_CACHE applies to updating directory's metadata items.

For different applications, MDS will authorize Clients different cache permissions (or cache modes) of the metadata items cached. For those directories and files accessed frequently, allowing Clients to cache their metadata items will cause MDS sending many messages to tell Clients the metadata item they cached is invalid, which will lead to much network overhead, so Clients can only cache the metadata item in NO_CACHE mode. For applications similar to web accessing and video demanding, users commonly do only read operations, and hardly do modify operations, so MDS can authorize Clients READ_CACHE cache permission for the metadata items. Cappella is a multi-user file system, if a user only do operations under his own directories, users are independent of each other, MDS can authorize Clients the WRITE_CACHE or CREATE_CACHE cache permissions, CLINETs can update metadata items cached locally, or create new directories and files locally, then commit the new metadata items back to MDS at a certain time, that can lower the overhead of communication between Clients and MDS.

3.3. Specific Implementation

According to the differences among the cache permissions, we define the priority among them is NO_CACHE< READ_CACHE< WRITE_CACHE< CREATE_CACHE. MDS authorizes different cache permissions of the metadata items according to their accessed situation.

When conflicts of accessing occur, MDS will execute corresponding callback functions to tell Clients cached the metadata items before to modify the cache permissions (commonly degrade the permissions), ensuring the consistence of metadata. In order to know the accessed situation of metadata, MDS records the number of receiving metadata requests and the times the function be called to ensure the cache's consistence, and calculates their ratio, if the ratio is bigger than 1, the metadata item's accessing is not shared frequently, or it is. The formula is as below:

AVG (the number of receiving requests for metadata) /AVG（the times of the function be called to ensure the cache consistency) > 1 (1)

For shared accessing, we separated shared reading from shared writing. Among the requests for the metadata item of the file or the directory, count the number of requests for reading metadata items(getattr) and modifying metadata items(setattr, mkdir, rmdir, create, unlink), calculate their ratio, if the ratio is bigger than 1, it's shared reading, or it's shared writing. The formula is as below:

AVG (the times reading metadata) /AVG (the times modifying metadata) > 1 (2)

After getting the information of shared accessing of metadata, the process of metadata cache permissions changing is showed in Figure 5.

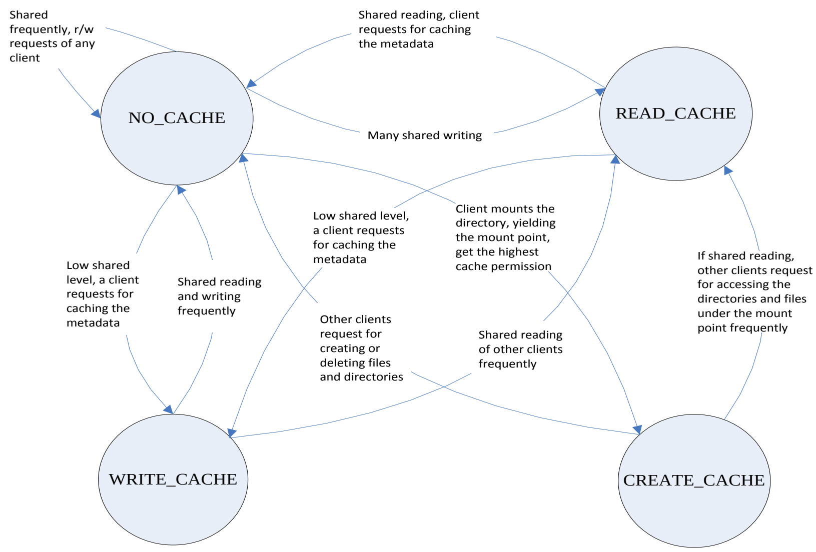

Figure 5. The Process of MDS Cache Permission Status Changing

MDS caches the Clients' information of each mount point, for the first Client C1 who mounts the mount point, MDS authorizes C1 the highest cache priority CREATE_CACHE, allowing it to execute a battery of operations like creating, deleting files and directories.

Following C1, if other Clients request the metadata of the files or directories under mount point, MDS will execute callback function, notifying C1 to commit all the updated metadata to MDS and set local cache invalid, metadata under mount point will be in NO_CACHE status for a moment.

Withdrawing the permissions for creating and deleting files and directories under mount point, MDS begins to response to C2 and other Clients. If Clients access the metadata in a shared reading way, they will all be authorized READ_CACHE cache permission; if many Clients create or delete frequently under this directory, they will be authorized NO_CACHE cache permission. For requests for metadata of files, if only a few Clients request for the metadata items, they will be authorized WRITE_CACHE cache permission; if the file is read by many users frequently, MDS authorizes Clients READ_CACHE cache permission; if the file is written by many user frequently, MDS authorizes Clients NO_CACHE cache permission.

For those metadata items in WRITE_CACHE status, when new requests arrive, MDS will notify the Client cached the metadata items to commit them back to MDS and set local cache invalid, assessing the shared situation, if the shared level is low, MDS continues to authorize new CLIDNTs WRITE_CACHE cache permission. If it was shared reading, MDS authorizes new Clients READ_CACHE cache permission; if it's shared writing, MDS authorizes new Clients NO_CACHE cache permission.

For those metadata items in READ_CACHE status, we analyze the metadata of directories and files. If the metadata items belong to directories, in the situation of low shared level or shared reading, MDS continues to authorize new Clients READ_CACHE cache permission; if the metadata items are shared-written frequently, MDS will authorize new Clients NO_CACHE cache permission in order to avoid messages to notify Clients to set their cache invalid. If the metadata items belong to files, and also if the shared level is low, MDS continues to authorize new CLIDNTs WRITE_CACHE cache permission. If it was shared reading, MDS authorizes new Clients READ_CACHE cache permission; if it's shared writing, MDS authorizes new Clients NO_CACHE cache permission.

From the process of authorizing cache permission, CREATE_CACHE can only be authorized to the Client first mount the directory, when other Clients access the metadata in a shared way, the metadata items of the files and directories under the mount point can only change among WRITE_CACHE, READ_CACHE and NO_CACHE.

4. Experimental Test and Performance Analysis

4.1. Experimental Design and Procedure

The goal of the experiment is to verify the performance improving via the metadata cache policy proposed by this paper through testing. The experiment can be divided into two parts, one is that we test the cappella file system's IOPS before and after adding the metadata cache policy to see if the policy can improve the performance; the other is that we test cappella and luster respectively, compare their performance. At last, analyze the testing outcome, improve and optimize the metadata cache policy proposed in this paper.

The test tool we use in the experiment is mdtest, and test the IOPS of Directory creation/ Directory stat/ Directory removal/ File creation/ File stat/ File removal/ Tree creation/ Tree removal these operations and so on.

Experimental environment is as below:

(1) Hardware environment：2x CPU(Intel Xeon E5560, 4x cores (each 2.8GHz)); 36GB DDR3 RAM; 8x SAS Disks (15000RPM, 146GB) configured RAID5 （7DISKs）; Mellanox InfiniBand QDR 40Gb/s NIC

Switch: Mellanox 36ports at 120Gb/s InfiniBand Switches

(2) Software environment：OS: RedHat Enterprise Linux 5.3 (Linux-2.6.27.5-x86_64)；

(3) Test parameter: The number of Clients grows from 1 to 5, and the number of MDS and OSD are both 1.

4.2. Experimental Result and Analysis

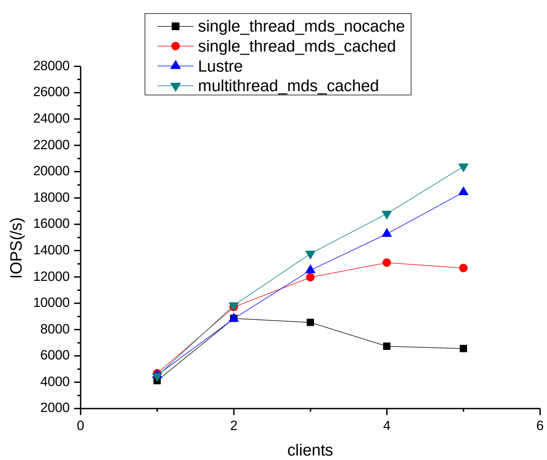

Figure 6. Directory Creation IOPS Comparison

Figure 6 shows, when the number of Client is 1 and 2, for the single thread MDS, there are hardly any differences between the versions with metadata cache and without metadata cache, and the performance is still growing. With the number of Clients growing, the performance of MDS with metadata cache is growing slowly, when the number of Clients is 4, the performance reaches peak, and then IOPS don't grow. For the MDS without metadata cache, when the number of Clients is more than 2, the performance will decrease. For multi-thread MDS with metadata cache and luster, the performance of both is growing, and Cappella is much better than luster.

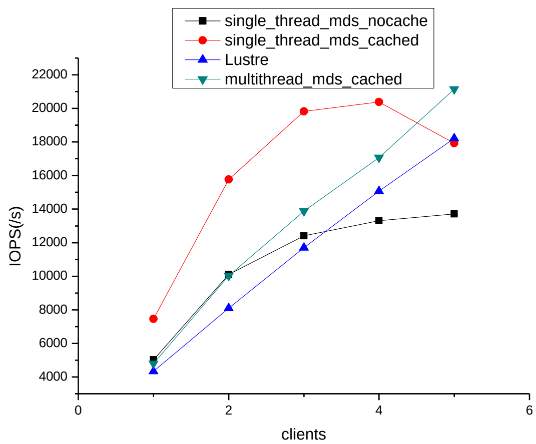

Figure 7. Directory Removal IOPS Comparison

Figure 7 shows, the speed of rmdir operation in MDS with metadata cache is twice as fast as in MDS without metadata cache, for multi-thread MDS with metadata cache and luster, the performance of both is growing, and Cappella is much better than luster.

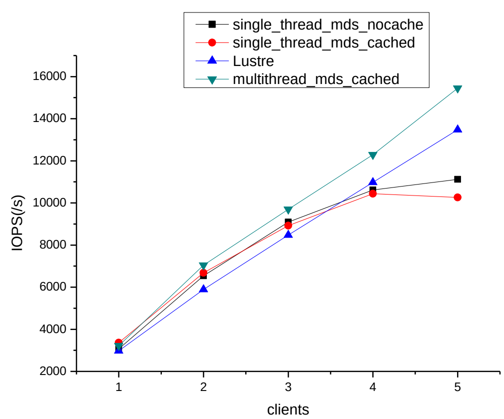

Figure 8. File Creation IOPS Comparison

Figure 8 shows, for all systems, there is a linear growth in file creating operation rate with the number of Clients growing, multi-thread Cappella with metadata cache management is much better than luster. Now Cappella file system only support READ_CACHE mode, the single-thread MDS with cache management and that without cache management have the same performance growing situation, when the number of Clients is more than 4, IOPS don't grow any more.

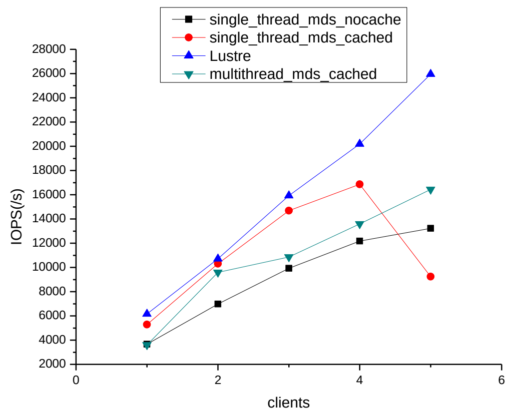

Figure 9. File Removal IOPS Comparison

Figure 9 shows under the same condition, after supporting metadata cache management, the IOPS of file deleting is improved much. MDS can support multiple threads and metadata cache both, that means it has better expansibility.

From the test outcomes above, for all operations the IOPS of Cappella is much better than that of luster except file removing operation. The performance of MDS with metadata cache is 2-3 times as good as that of MDS without metadata, and the performance of single thread MDS will reach a peak with the number of Clients growing, but the performance of multi-thread MDS will continue to grow, it has better expansibility.

5. Conclusions

This paper does a research on the cache technologies used on global distributed file system, combines with multi-user parallel file system cappella that we developed, and propose a new metadata cache management policy. The policy is easy and efficient, not only improves the performance of MDS, decreases the user's delay, but also controls the consistence of metadata cache. Experiment shows the performance of MDS with metadata cache is 2-3 times as good as that of MDS without metadata cache. According the tests in the same environment, the performance of Cappella is much better than that of luster.

Acknowledgements

This project was supported by the National Basic Research Program of China (973 Program) under Grant No.2011CB302301, New Century Excellent Talents in University NCET-04-0693 , the National Science Foundation of China No.60703046 and No.60873028, the National High Technology Research and Development Program(863 Program) of China under Grant No.2009AA01A401, 2009AA01A402, Changjiang innovative group of Education of China No. IRT0725.

References

[1] J. L. Zhang and F. Dan, "Massive Information Storage", Science Publishing House, **(2003),** pp.1-124.
[2] M. Mesnier, G. R. Gander and E. Riedel, "Object-based Storage", IEEE Communications Magazine, **(2003),** pp. 84-91.
[3] M. D. Dahlin, R. Y. Wang, T. E. Anderson and D. A. Patterson, "Cooperative Caching: Using Remote Client Memory to Improve File System Performance", First OSDI, **(1994),** pp.267-280.

[4] G. A. Gibson , D. F. Nagle, K. Amiri, F. W. Chang, E. M. Feinberg, H. Gobioff , C. Lee, B. Ozceri , E. Riedel, D. Rochberg and J. Zelenka, "File Server Scaling with Network-Attached Secure Disks", Performance Evaluation Review, **(1997),** pp. 272-284.
[5] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller and D. D. E. Long, "File System Workload Analysis For Large Scale Scientific Computing Applications", The 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies, **(2004)** April, College Park, Maryland, USA.
[6] H. Cho, S. Kim and S. Lee, "Analysis of Long-Term File System Activities on Cluster Systems", World Academy of Science, Engineering and Technology, vol. 3, **(2009).**
[7] P. J. Braam, "The Lustre Storage Architecture", Cluster File Systems, Inc. Whiter Paper, http://www.clusterfs.com, **(2003).**
[8] D. Hitz, J. Lau and M. Malcom, "File System Design for an NFS File Server Appliance", Proceedings of the Winter 1994 USENIX Conference, **(1994),** San Francisco, California, USA.
[9] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel and D. Hitz, "NFS version 3: Design and implementation", Proceedings of the Summer 1994 USENIX Conference, **(1994)** June 6 - 10, Boston, USA.
[10] N. N. Michael , B. W. Brent and K. O. John, "University of California at Berkeley Caching in the Sprite Network File System", ACM Transactions on Computer Systems, **(1988),** pp. 134-154.

Authors

Jingning Liu, has more than 20 years of experience of research and education in field of computer and applications. She is a professor at Huazhong University of Science & Technology and focuses on the teaching of the application technology of computer. Her research interests include grid/cloud computing and advanced cloud applications.

Junjian Chen, received his Ph.D. degree in computer science at Huazhong University of Science & Technology, majoring in Science of Computer. Now, he is a lecturer in Foshan University. His research interests include cloud computing and distributed system.

Wei Tong, is a lecturer in Huazhong University of Science & Technology and focuses on the teaching of computer architecture as well as the application technology of cloud storage.

Lei Tian, received his Ph.D. degree in computer science at Huazhong University of Science & Technology. His research interests include computer architecture and distributed system.

Cong Chen, is a Postgraduate student in Huazhong University of Science & Technology. His current research interests include cloud storage and cloud computing.

