Memory based metadata server for cluster file systems

Jing Xing
Institute of Computing Technology,
Chinese Academy of Sciences
Key Laboratory of Computer Architecture and System,
Chinese Academy of Sciences
Graduate University of Chinese Academy of Sciences
Beijing, China
xingjing@ncic.ac.cn

Jin Xiong, Jie Ma, Ninghui Sun
Institute of Computing Technology,
Chinese Academy of Sciences
Key Laboratory of Computer Architecture and System,
Chinese Academy of Sciences
Beijing, China
{xj, majie, snh}@ncic.ac.cn

***Abstract*— n high performance computing environment, the metadata servers of distributed file system become critical to impact overall system performance. An approach of memory based metadata server is proposed, instead of the disk based approach. We present a metadata management system with matrix organization, non-overhead reliable mechanism and static scalability method, which is design to efficiently utilize large memory and provide high performance. We examine and demonstrate the performance, overhead of reliability and scalability in a test bed environment of 28 machines. The result shows that the performance of our system is higher than other traditional distributed file system, the reliability can be achieved with little overhead and the metadata servers can be linear scaling.**

Keywords- metadata server, cluster file system

I. Introduction

In high performance computing environment, the performance of metadata of file system is critical [6]. As an example, the large web server and mail server has millions of users, they need to handle hundreds of thousand metadata operations every second. In the Accelerated Strategic Computing Initiative (ASCI) projects, the requirement is 6000-15000 files/second metadata processing power in 2005, 18000-450000 files/second in 2008, increasing about 3 times every three years [1].

In large distributed storage systems, decoupling metadata processing from file block operations is widely used. In such systems, the metadata server (MDS) maintains file system namespace and the storage servers process the read and write requests. Although the size of metadata is relatively small compared with the overall size of the file system, the metadata operation may make up over 50% of all file system operations [14], making the MDS critical to overall system performance. As the metadata is highly interdependence, MDS is hard to scale easily.

In general, MDS cluster is used by large storage environment to achieve high metadata performance. For example, ceph supports more than 250,000 metadata operations per second by using a 128-node MDS cluster [13]. The scale system achieves high performance with high cost: Each metadata server only contributes 2000 ops per second.

In metadata operations, disk access accounted for a large part of overhead. If there is no disk access in the metadata operations, the performance of MDS will be improved greatly. Usually, MDS only stores recent-used metadata in memory. This makes sense when memory is far more expensive than disk. Currently, it becomes feasible to store all the active metadata in memory. The memory based metadata server holds all the active metadata in memory and there will be no disk access in transaction processing. It can provide faster response and higher throughputs, which is especially important for real time applications where metadata request have to be completed by their specified deadlines.

Memory clearly has different properties from that of magnetic disks, and these differences have profound implications on the design and performance of our approach. The key issues caused by these differences are described below:

1) Memory access is an order of magnitude faster than disk access. As we hold all the active metadata in large memory, how do we organize metadata and improve the utilization of memory?

2) Memory is volatile, while disk storage is not. Metadata in memory need to be written to the disk in period to keep the content safe. How do we design the reliability mechanism to interleave with metadata processing approach to not affect the performance?

3) Disk holds far more content than memory which cost the same money. How to process more metadata than the memory limitation?

This paper addresses the above issues. More specifically, it has the following contributions:

- We have studied the effect of using memory as the persistent storage of metadata, and then proposed the new metadata organization and processing approach to get full use of memory.

- We present a reliability mechanism without affecting performance. Metadata in memory will be safe even when the system crushes.
- We supply a partition method to eliminate the capacity limitation. Metadata can be processed as much as ordinary MDS.

The rest of this paper is organized as follows: Section 2 presents the design issues. Detailed implementation is described in section 3. Experiments and results are discussed in section 4. Relevant research work is summarized in section 5. And finally we conclude our work in section 6.

II. Design issues

A. Metadata Organization

There are two premises in metadata organization design. One is that although memory capacity increases rapidly, it is still limited compared with the disk capacity. It is necessary to switch metadata between memory and disk. Another is that memory capacity is enough compared with the active metadata set. In a time t, the total active metadata set is limited. If memory capacity is larger than the active metadata set, there is no disk access in time t after all the active metadata set is loaded into the memory. If we partition the metadata appropriately, and load the metadata partition into memory when needed, memory can be unlimited for users.

As memory is volatile, memory backups will have to be taken frequently to keep the metadata consistency. Writing back a metadata partition at one-time can achieve good consistency and simplicity, but this backup mechanism will block the normal process and lead to poor performance. If we divide each metadata partition into multiple boxes which contains the same number of metadata, and write back one box at one-time, then the backup mechanism will achieve high disk bandwidth utilization and interleave with the normal operation.

Metadata includes attribute information and namespace information. In ordinary file system, they are organized in different ways to utilize disk locality. However, as the two metadata information of a metadata will be in or out memory together in our approach, we combine the two metadata information, and name it as combined metadata. Once we get the combined metadata of a file, we can use all the metadata information of the file. As the access to combined metadata is atomically, system crush would not lead to inconsistency of the two kinds of metadata information.

B. Reliability

As memory is gigabyte-scale in a metadata server, it is impossible to write back the whole memory every time. We can just write back dirty metadata boxes to accomplish memory backup. Log recording is needed to keep the consistency of the whole metadata.

In large distributed file system, there may be tens of thousand metadata transactions committing to MDS every second, and large number of log record will be committed to disk. To improve the speed of log record committing, we adopt group commit method to relieve the log bottleneck. However, using one log buffer will still block the system when the full log buffer is written to disk. We adopt two log buffers to deal with this problem. When one log buffer is being written to the disk, another free buffer can receive the following log record.

C. Scalability

In large scale distributed storage environments such as supercomputer centers and large web service centers, users generally paid for a private storage space and shared CPU time. User's private files are stored under their own directory and these files are not allowed to be accessed by others. The partition method to divide namespace into multiple parts can be used to improve the scalability. Each metadata partition is a sub-tree of the whole namespace. Multiple metadata partitions can be distributed in several metadata servers. Since metadata partitions are relatively independent, they can scale well without much overhead.

To maintain the consistency of partition information, one of the metadata servers is selected as master server to control the overall partition information. All the partition operations need to be processed through master server.

In the MDS cluster environment, client needs to know the mapping between metadata and metadata server. However, maintaining the mapping of hundreds of millions entries will be costly and inefficient. We can hide partition information in metadata information, then maintaining the mapping between partition and metadata server at client is enough.

III. Implementation

We implement the memory based metadata server in our DCFS3, which is the cluster file system of 200Flops Dawning 5000A system.

A. Matrix Metadata Organization

Each metadata partition can be treated as a memory file system when it is loaded into the memory. The partition is organized as a large array of metadata boxes. The number of boxes in the partition depends on the partition size. Every metadata box contains fixed number of combined metadata. A partition can be seen as a matrix of combined metadata, as shown in Figure 1. With the partition ID and the box ID which a combined metadata belongs to, we can get the address of the combined metadata and access it directly in memory.

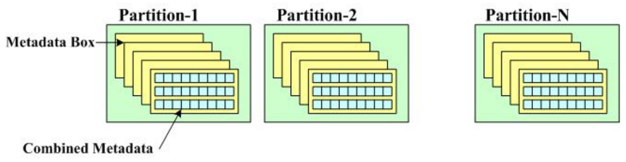

Figure 1. Metadata organization

In our approach, directory files do not exist at all. We keep the directory information in every combined metadata to maintain namespace information. Each combined metadata has four fields to record related directory information: one directory ID, two nearby siblings' ID and its first child ID if it is a directory. We can travel the whole namespace by using the information.

Traditionally, there are two modes to search metadata. One is to do lookup through pathname, and another is to get the metadata from ino. For name searching, we adopt hash table to index every combined metadata of a partition. Since every partition has limited number of combined metadata, each partition has its own hash table to do name searching. For ino searching, since we can locate a combined metadata easily by <partition ID, box ID， index>, we compose the ino of a combined metadata with the three location information.

B. *Reliability*

We use redo logging and memory backup to guarantee metadata safe. For redo logging, before a modified operation accomplished, a log record, which contains the operation parameters and the init status of metadata, must be committed to the log file. For backup mechanism, when start doing backup, dirty metadata boxes are written back to the disk. Memory backup is triggered by the time interval signal or the log mechanism.

Since the granularity of memory backup is metadata box, the consistency of overall metadata need be guaranteed by the cooperation of logging and memory backup. Two conditions need to be satisfied to make sure the cooperation succeed. Firstly, before write dirty metadata box to the disk, the log record related to the metadata need to be committed to disk. Secondly, we can not release a log record in disk until the metadata which was referred to has already been written to disk.

We use group commit and double log-buffer policy in the logging mechanism to speed up log recording. Under group commit, the records of several transactions are accumulated in a log buffer. The log buffer is flushed to the disk in a single disk operation. For double log buffer policy, when one log buffer is being written to the disk, another free buffer can receive the following log record without blocking log record committing. In this policy, it blocks the log system only when both of the two log buffers are full to be written to disk. And this situation happens only when the speed of log record committing is faster than disk access. Support that the disk bandwidth of SCSI disk is 70MB/s, and the average size of log record is 0.5KB. The log system with single SCSI disk can support committing 140 thousand records per second without block normal operation. Since this number is higher than the peak performance of single metadata server, there will be no log blocking.

We use the granularity of metadata box to write back. When writing back metadata box to disk, operations refer to the box can not be processed until the backup operation finished. We use read-write lock on each box to control the access to box. Before backup thread writes a metadata box to disk, it needs to get the write lock of the box to prevent other threads access. When a service thread wants to access a metadata, it needs to get the read lock of the box which the metadata refers to. Multiple service threads can share the read lock of a metadata box at the same time, and backup thread can not get the write lock of the box until all the service threads release the read lock.

If a metadata server crushes, the recovery demon asks the master server for new location of every partition in the crushed server. After the master server assigns these partitions to other metadata servers, these partitions are recovered.

C. *Scalability*

Every partition has a global ID and some related information to record its status. The information is maintained by the master server. Master server also collects the information such as CPU utilization and free memory from other metadata servers. Based on the resource information, master server can balance workload among MDS cluster. When a user ask master server to load a partition, master server will choose the one with largest free memory. If there is no metadata server can hold the request partition, failed reply will be send to the user.

In MDS cluster environments, client needs to maintain the mapping between metadata partition and metadata server. When client is mounted, the mapping table is initialized to be empty. When searching result returns empty or error, client will ask master server for right mapping information. Since every client only needs to update mapping information when partition location changes, the mapping request will not affect master servers performance.

IV. EVALUATION

We evaluate the DCFS3 with our approach in a test bed of 28 servers by benchmarks.

A. *Experiment Configuration*

Each server in the test bed is a server with two 2.2GHz AMD opteron CPUs, 2GB ECC DDR SDRAM, and two 146 GB SEAGATE hard drivers. Each server uses a single Intel PRO/1000 gigabit Ethernet adaptor to connect to a Foundry FASTIRON superX gigabit switch. We designate four servers as object storage devices, four servers as metadata servers and the rest as clients. The DCFS3 can be accessed through VFS interface at client. It obeys POSIX semantics and can be used as local file system.

B. *Performance*

Initially, we evaluate the performance with different metadata management approaches. PVFS2 and Lustre are selected to compare with DCFS3. Since PVFS2 and Lustre are single metadata architecture, we configure a testing system with one metadata server, four storage server and twenty clients. To make it equal for the other two systems, we limited the working set to be hold in memory. MDTEST benchmarks are used to understand the metadata performance.

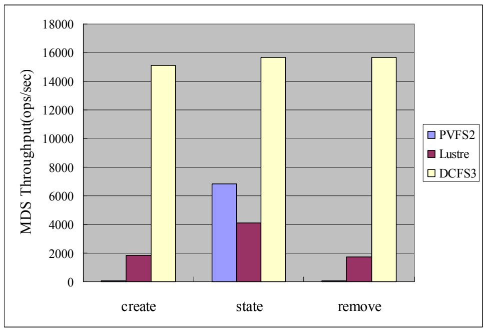

Figure 2. Performance

In this benchmark, twenty clients submit request to the metadata server concurrently. These clients first create 200,000 files, then state these files, and remove these files at last. Figure 2 shows the performance result. For create operation, the throughput of DCFS3 is about 8 times more than Lustre and about 213 times more than PVFS2. For state operation, the throughput of DCFS3 is about 4 times more than Lustre and about 2 times more than PVFS. For remove operation, throughput of DCFS3 is about 9 times more than Lustre and 290 times more than PVFS2. Since PVFS2 adopt database to manage metadata, it has the lowest performance when doing modify operation. Lustre also shows weak result as it uses traditional method which can not utilize memory well.

C. *Reliability Overhead*

The overhead to backup metadata and keep the consistency of system is critical. We compare reliable and unreliable policy to show the overhead of reliability mechanism.

We use the MDTEST benchmark to produce workload with the working set from 100,000 to 1,000,000. Twenty clients are used to submit request and two servers are configured as the metadata servers. Under each working set, the MDTEST benchmark will repeat several times to make sure that the memory back is taken. The result of the benchmark will not be recorded until memory back is taken.

Figure 3 show that the throughputs of the two polices are almost the same, although larger working set will cause more metadata to be written back to disk. The overhead of reliable mechanism can be neglected.

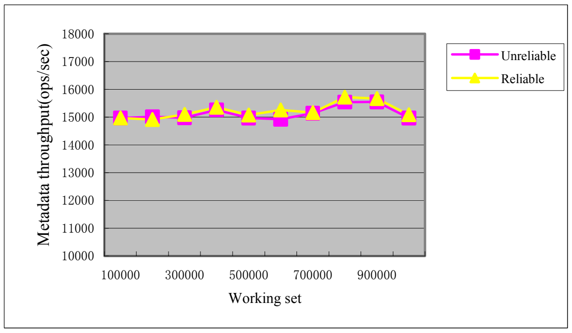

Figure 3. Reliability Overhead

D. *Scalability*

In this section, we evaluate the scalability using a micro-benchmark which creates flat directory structure. In the benchmark, each client has several threads to submit create file request to multiple metadata servers. 200,000 files are equally created at every metadata server. Each metadata server only loads one metadata partition.

Figure 4 plots overall through-put(y) versus total working thread number(x) for a 1-, 2-, and 4-node metadata servers under the micro-benchmark. Metadata server can not reach the peak performance until the thread number is more than 40. The throughput of MDS cluster can be linear scaling when each metadata server is running at full capacity.

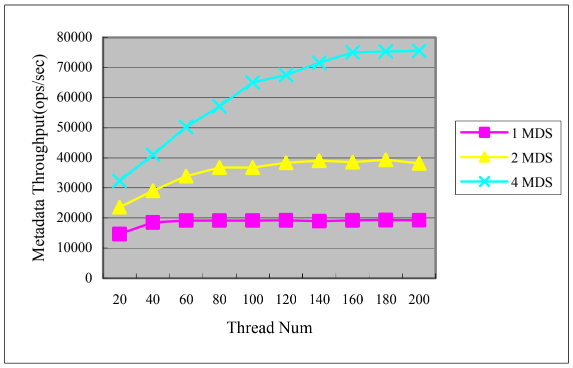

Figure 4. Scalability

V. RELATED WORKS

Local file system focus on utilizing disk locality as disk access accounts for a large part of metadata operation overhead [3, 4, 10, 11, 12]. In these systems, memory is used as cache of disk using LRU to replace metadata in memory.

There are many research works on building MDS for distributed file systems [2, 5, 13, 15]. Ceph is designed for petabyte-scale distributed storage. It utilizes a dynamic sub-tree partitioning strategy to distribute workload while maximizing overall scalability. It also uses embedded inodes to exploit the locality of reference and to simplify storage. Although their embedded inode is similar to ours, a key difference is that our embedded inode is organized without directory limitation. The location of embedded inode is depended on their creating time. In memory utilization, ceph cache prefix inodes for all items in the cache, which means that directory inodes may not be removed until items contained within them are expired. Though prefix inodes has the effect of decreasing the cache hit rate, its effect is related to the distribution of requests throughout the file system. Prefix caching dose not work well when leaf items of directory are mainly accessed.

Google file system uses a single master to maintain all file system metadata [8]. All metadata is kept in master's memory. Google file system does not have a per-directory data structure. It does not support POSIX semantics as the lack of directory structure. It logically represents its namespace as a lookup table mapping full pathnames to metadata. Using table to manage metadata will be simple and easy to implement, and works well when there is little delete operation. However, as table is a linear structure, it will be difficult to support changing namespace dynamically. The master utilizes snapshot and log to keep consistency of metadata, which is similar with our log and backup mechanism. The difference is that we update the original metadata in disk continually and Google file system switch metadata view in disk among different snapshot.

Memory resident database systems (MMDBs) store their data in main memory and provide very high-speed access [7, 9]. In contrast to conventional database systems, which are optimized for the particular characteristics of disk storage mechanisms, memory resident systems use different optimizations to structure and organize data, as well as to make it reliable. Although database is different with metadata server, there are a lot in common in the fields of index searching, concurrency control, commit processing, data representation, recovery mechanism and so on. However, as metadata is a tree structure which is different with the table structure of database, there are still differences in organizing memory space. Metadata still need to record its location in namespace tree in addition to the attribute information.

VI. CONCLUSIONS

In this paper, we propose a memory based metadata server to improve the performance. A matrix metadata organization is used to manage large memory space. Redo log and dirty box backup are used to provide reliability without much overhead. Static partition is used to hold active metadata in memory for scalability.

The performance test shows that this approach has much better performance compared with Lustre and PVFS2. The overhead of keeping reliability can be neglected. The system can linearly scale up.

In the future, the more experiments will be done on the real large scale Dawning5000A's system. And the performance of real applications is also needed to be emulated.

ACKNOWLEDGMENT

The work is supported in part by the National Natural Science Foundation of China under grant numbered 60633040 and 60573099, and the National High-Tech Research and Development Program of China under grant numbered 2006AA01A102.

REFERENCES

[1] Statement of Work:SGS File System DOE National Nuclear Security Administration and the DOD National Security Agency, http://www.lustre.org/docs/SGSRFP.pdf

[2] A. R. Butt, T. A. Johnson, Y. Zheng, and Y. C. Hu. Kosha: A peer-to-peer enhancement for the network file system. Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, 2004.

[3] R. Card, T. Tso, and S. Tweedie. Design and implementation of the second extended filesystem. Proceedings of the First Dutch International Symposium on Linux, 1994.

[4] M. Corporation. Fat: General overview of on-disk format. Hardware White Paper, 2000.

[5] J. R. Douceur and J. Howell. Distributed directory service in the farsite file system. Proceedings of the 7th symposium on Operating systems design and implementation, pages 321–334, 2006.

[6] J. F. Gantz, D. Reinsel, C. Chute,W. Schlichting, J. McArthur, S. Minton, I. Xheneti, A. Toncheva, and A. Manfrediz. The expanding digital universe: A forecast of worldwide information growth through 2010. IDC white paper Csponsored by EMC, 2007.

[7] H. Garcia-Molina and K. Salem. Main memory database systems: an overview. IEEE Transactions on Knowledge and Data Engineering, pages 509–516, Dec. 1992.

[8] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 29–43, 2003.

[9] K. Li and J. Naughton. Multiprocessor main memory transaction processing. Proceedings of the first international symposium on Databases in parallel and distributed systems, 1996.

[10] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for unix. Technical Report: CSD-83-147, 1983.

[11] D. Phillips. A directory index for ext2. Proceedings of the 5th annual conference on Linux Showcase and Conference, page 20, 2001.

[12] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the xfs file system. Proceedings. of the USENIX 1996 Annual Technical Conference, 1996.

[13] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: a scalable, high-performance distributed file system. Proceedings of the 7th symposium on Operating systems design and implementation, pages 307–320, 2006.

[14] S. A.Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller. Dynamic metadata management for petabyte-scale file systems. Proceedings of the 2004 ACM/IEEE conference on Supercomputing, page 4, 2004.

[15] Y. Zhu, H. Jiang, J. Wang, and F. Xian. Hba: Distributed metadata management for large cluster-based storage systems. IEEE Transactions on Parallel and Distributed Systems, 2008.