A Novel Collaborative Fault-Tolerance Mechanism for Metadata Server Cluster

Z.X LI, Q.S WEI, Y.H WANG

Data Storage Institute
Singapore

{LI_Zhixiang, WEI_Qingsong, WANG_Yonghong@dsi.a-star.edu.sg}

***Abstract*- In object-based storage system, metadata residing in Metadata Server (MDS) is crucial to the survival of overall system. Keeping metadata always available is becoming one of the key issues to be addressed for object-based storage system design and implementation.**

This paper presents a novel cooperative fault-tolerance mechanism for metadata server cluster, called Co-MDS. The Co-MDS consolidates disk space of individual MDS into a global metadata storage space and provides high fault-tolerance by distributing metadata and its parity data across collaborative metadata server cluster environment. With this mechanism, metadata is still available even when a part of metadata servers fail. Our design is transparent for applications and complies with OSD T10 standard. The experiments demonstrate that the proposed collaborative mechanism can offer high fault-tolerance and significant performance.

I. INTRODUCTION

Three storage architectures in common use today are direct-attached storage (DAS), storage area network (SAN), and network-attached storage (NAS). Today's architectures put challenge for system designers to decide on what aspect is most important to the customer environment, practical selecting procedure involves a trade-off. DAS connects block-based storage devices directly to the host and offers high performance and minimal security concerns, but suffers from the limitation on distance, scalability, and data sharing. NAS uses dedicated file server to provide cross-platform file-based access to storage. However, the performance of NAS will downgrade as the number of requests grows. SAN uses a switched fabric to connect storage devices and promises inexpensive and shared access to centrally-managed storage. But in existing SAN deployments, the storage is partitioned among hosts, and the hosts treat the storage as if it were directly attached. The ability of the servers to share data over the SAN is limited. As such, next generation storage architecture, Object-Based Storage architecture, is proposed to provides both the direct access nature of SAN and the heterogeneous data sharing of NAS [1,2,3].

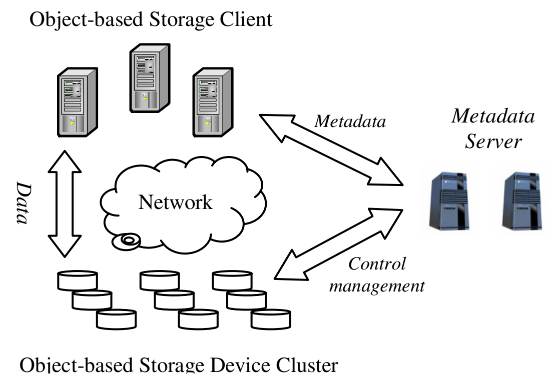
Object-based Storage Client
Metadata
Metadata Server
Data
Network
Control management
Object-based Storage Device Cluster

Figure 1: Object based Storage System Architecture

Object-based storage system consists of three main components: MDS cluster, Object-based Storage Device (OSD) cluster and Object-based Storage Clients (OSC, namely Application Servers), attached by network, shown as figure 1. The MDS presents a global name space for object location and secure access. File is stored across OSD clusters in form of objects. When a client requests for a file, application server first needs to get access capacity and the metadata from MDS. Once this is done, object data is transferred directly between the application server and the OSD cluster in parallel way [4,5].

Even though object-based storage system has the potential to offer high performance, it cannot be realized unless MDS is available. The metadata residing in MDS is crucial to the survival of overall system. Thus, keeping metadata always available is becoming key issue to be addressed for object-based storage system. High fault-tolerance of metadata or MDS is desirable, and therefore, it is very important to develop a high fault-tolerance storage mechanism for MDS.

RAID [6,7,8] is the most widely used technology to ensure the data reliability, but it has following shortages: 1. It only allows one disk failure (RAID 6 allows two disks failure at best but its disk utilization is much low.), if there are two or more disks failure, data will be unavailable. 2. The distance RAID can reach is very limited. It is unable to realize long-distance distributed redundant storage. In addition, the fault-tolerance of SAN still relies on RAID [9,10]. 3. Some organization in the world is conducting research on iRAID but the concept is still limited to the traditional RAID and no breakthrough is made yet [11].

In this paper, we addresses above issues by designing a novel cooperative fault-tolerance mechanism for MDS, called Co-MDS. Using the disk space of MDS nodes, it constructs a single metadata storage space to which applications can access with OSD-SCSI commands. In Co-MDS, metadata is stored in form of Original Fragments and their Parity Fragments among the MDS nodes of the cluster. Metadata is still available and accessible from the global metadata storage space even when a part of MDS nodes are offline or crashed. In addition, through parallel metadata transfer mechanism, Co-MDS can provide high-performance in term of metadata read and metadata write.

The rest of this paper is as follows. Section 2 provides an architecture overview of Co-MDS. In Section 3, we present the detailed design and implementation. Evaluation and test result is presented in Section 4. Finally, Section 5 summarizes the paper with conclusion and future works.

II. Architecture of Co-MDS

Each node in a MDS cluster is not only a member of the cluster, but also an independent computer that has its own memory and disk space. In this paper, we utilize individual disk space of each MDS node in the cluster to construct a public metadata storage space to which applications can access as they access local data. Figure 2 shows its architecture.

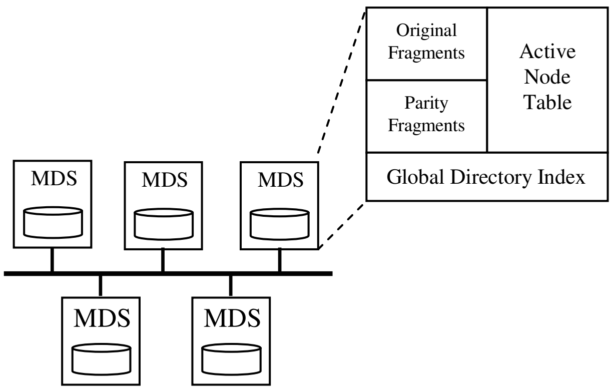

Figure 2. Architecture of Co-MDS

In the Co-MDS, Global Directory Index (GDI) manages global metadata storage space where storage information of metadata is preserved. Each peer on the Co-MDS has a copy of GDI and takes part in GDI synchronization process by which GDI information of each peer is synchronized and the consistency of global data is maintained.

Active Node Table (ANT) keeps a list of all active metadata servers. With MDS joining and exiting the cluster, system adds and deletes corresponding entry in the ANT to reflect the change of environment.

As the application writes metadata to the global space, the system firstly queries the ANT and selects the Available MDS Array (AMA) according to the status of active MDS nodes such as workload and free disk space, then strips the metadata into multiple fragments and calculates their XOR parity fragments, finally distributes these original fragments and parity fragments in different metadata servers selected by AMA and refreshes the GDI accordingly.

When an application requests for a file, the system queries the GDI to obtain the location information of original fragments and checks fragments of the file and the set of storing hosts which store these fragments, selects the set of available storing hosts by intersection with the ANT, then gets fragments from the online storing hosts and reconstructs the file.

III. The Design of the Co-MDS

A. *Global Directory Index and its Synchronization Mechanism*

In order to realize a single namespace, every MDS holds a Global Directory Index (GDI) to maintain the information of all metadata stored in the global space. GDI keeps the information about how the metadata is stripped and where the original fragments and parity fragments are stored. The entity of GDI is analogous to the inode of Unix file system. Whereas the inode aggregates disk blocks on a single disk volume to compose a file, GDI aggregates original fragments and parity fragments from different metadata server to compose a metadata [3]. By querying the GDI with file name, Co-MDS can know how the metadata of the file is stripped and which MDS stores the original fragment or parity fragment.

Co-MDS is a P2P system, in which every MDS has a copy of GDI. In order to maintain the consistency of GDI, each MDS takes part in the information synchronization process. GDI utilizes Event Driven Synchronization Mechanism (EDSM) to synchronize the GDI of all metadata servers so that the consistency of global data space can be ensured.

EDSM method means when either of the events of reading/writing or timer expiration occurs, EDSM drives MDS node to inform all other MDS nodes of the system to update GDI with synchronization messages. If there is no event occurring in a certain period of time, expiration of timer will activate EDSM.

There are two kinds of EDSM: SYN-ANS synchronization and SYN-NOANS synchronization. SYN-ANS synchronization requests both the receiver and the sender to update its local GDI synchronously and is used by the sender to push each node on the MDS cluster to synchronize its GDI, as shown in Figure 3 (A). The sender sends SYN-ANS message SYN (GDI, ANS) to all other online nodes and waits for answers from receivers. When a receiver gets such synchronization message, it updates its GDI and sends SYN-NOANS message SYN (GDI, NOANS) back to the sender.

When the sender gets the SYN-NOANS message from the receiver, it also updates its GDI accordingly [4].

Unlike SYN-ANS, SYN-NOANS only needs each receiver to update its GDI, and the sender is not required to update its GDI synchronously. SYN-NOANS synchronization is used to inform other online MDS nodes to synchronize its GDI when a MDS node succeeds in writing, as shown in Figure 3(B). The sender sends SYN-NOANS message SYN (GDI, NOSYN) to all other online MDS nodes. When a receiver gets this synchronization message, it updates its GDI immediately.

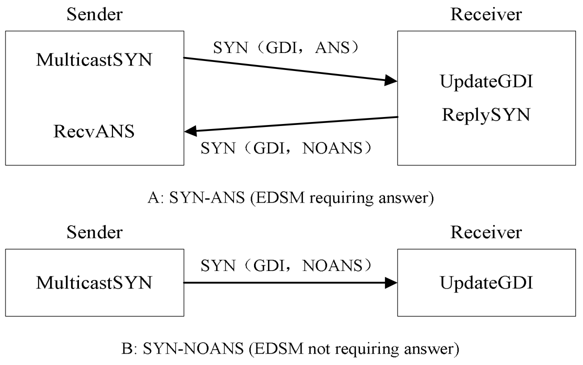

Figure 3: GDI Event Driven Synchronization Mechanism (EDSM)

B. *Active Node Table and its Synchronization Mechanism*

In Co-MDS, each node maintains an Active Nodes Table (ANT) which keeps a list of active MDS nodes in the system, including IP address of each node, free disk space and time stamp. IP address is unique physical position identifier in the Co-MDS, free disk space is used to choose suitable node for metadata writing, and time stamp is the last updated time of ANT.

When an application writes metadata to global space, the system queries the ANT firstly, selects available node array according to the network status and free disk space of active nodes and then decides how to strip the metadata. When reading metadata from the global space, the system gets the list of available nodes according to GDI and ANT and reads metadata fragments from the selected nodes. ANT plays an important role in metadata writing and reading.

Because of the importance of the ANT, it is necessary to maintain the consistency of ANT. This means when new node joins the system, each online node should add it to its own local ANT, when a node exits the system either normally or abnormally, other online nodes have to be informed of this event and update its own ANT accordingly.

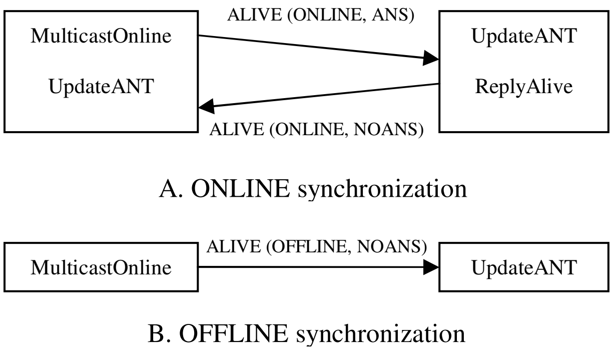

Figure 4: Active Node Table Synchronization Mechanism

Figure 4 shows the synchronization mechanism applied to ANT. There are two kinds of synchronization mechanism used in ANT: ONLINE synchronization and OFFLINE synchronization. ANT ONLINE synchronization needs senders and receivers to update their local ANT synchronously. As shown in Figure 4(A), the sender sends an ONLINE message ALIVE (ONLINE，ANS) to all other online nodes when it joins Co-MDS system and waits for reply from receivers. When a receiver gets this message, it then updates its ANT and sends an answer message ALIVE（ONLINE，NOANS）back to the sender. When the sender receives the answer message from the receiver, it updates its ANT [5].

ANT OFFLINE synchronization mechanism is used to inform other online nodes to update their local ANT when a node exits Co-MDS system. It only needs the receiver to update its ANT without any reply. The sender is not required to update its ANT because it is going to be offline soon. As shown in Figure 4(B), when a node exits the system, it sends an OFFLINE message ALIVE（OFFLINE，NOANS）to all other online nodes. When a receiver gets this message, it deletes the sender from its ANT. If a host quits abnormally, the periodic ANT synchronization messages sent by other nodes will ensure that the crashed node be detected and deleted from the ANT of each online node timely.

C. *Metadata Writing*

When an application writes metadata to the global space, the system will strip the metadata into multiple fragments and calculate XOR parity fragments and store these fragments in different nodes in the cluster. The steps of metadata writing are explained as follows:

1) to query ANT to get the number of current active nodes denoted as M;
2) to calculate the number of nodes those are used to store the fragments according to striping algorithm, denoted as N (N<M);
3) to stripe the metadata into N_d original fragments, and obtain N_m XOR parity fragments through mutual XOR calculation among these N_d original fragments($N=N_d+N_m$);

4) to store the N_d original fragments and the N_m parity fragments into N selected online nodes;

5) to update GDI.

The goal is that even if some of the nodes those originally store some fragments of metadata become unavailable, complete metadata still can be retrieved from the rest of MDS nodes in Co-MDS system. In order to achieve the above goal, RPSA (Redundant Parity Striping Algorithm) is applied to Co-MDS system. The important principle of RPSA is that when an application writes a metadata to the global space, it stores not only the original fragments but also the redundant parity fragments. By doing so, one can use the parity fragments to recover the whole metadata even when some of the original fragments are not available. RPSA can be described as the following formula (1):

$$\begin{cases} N = n + C_n^2 & \{n \mid n \geq 2, n \in Z\} \\ N_n \leq M \leq N_{n+1} & (M \geq 3) \end{cases} \quad (1)$$

In formula (1), where M denotes the number of active nodes; n denotes the number of original fragments; C_n^2 represents the number of parity fragments; N denotes the number of the selected nodes storing original fragments and parity fragments.

When an application writes metadata, the system calculates original fragments number n through active nodes number M, then obtains the number of storing nodes N, selects N nodes from the M active nodes, strips the metadata into n original fragments and obtains C_n^2 parity fragments through mutual XOR calculation, puts the n original fragments and the C_n^2 parity fragments into the N selected storing nodes respectively, finally updates GDI.

For example, if the number of online nodes M is 7, according to RPSA, the system obtains n=3 and N=6 firstly, selects 6 storing nodes, divides the metadata into 3 fragments, calculates their mutual XOR parity, stores these 6 fragments in the 6 selected storing nodes, and finally updates GDI. Figure5 shows the whole process.

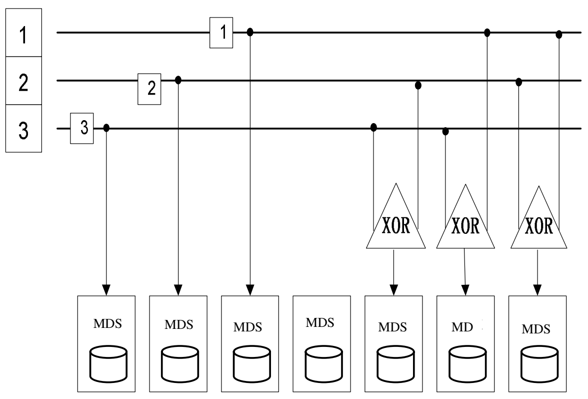

Figure 5: Metadata writing (7 active MDS Nodes, 6 MDS Nodes are selected to store metadata)

D. Metadata Reading

When an application reads a metadata from Co-MDS, it needs to get enough corresponding fragments from the MDS nodes and congregate them into an integrated metadata. The steps of metadata reading are as follows:

1) to query GDI with file name to obtain the storage information of the metadata (original fragments and parity fragments);

2) to get the set of storing nodes and make intersection with ANT to obtain the set of available storing nodes;

3) to retrieve the original fragments or XOR parity fragments from available storing nodes selected;

4) to congregate these fragments into integrated metadata [6].

When reading a metadata, in order to reduce the overhead of calculation and speed up the reading, Co-MDS uses the following strategy: reading parity fragments only when the metadata can not be reconstructed from original fragments. If some original fragments are unavailable, Co-MDS can obtain all the original fragments through XOR calculation and finish the metadata reconstruction successfully.

Considering the worst case, when the number of online nodes is Cn2+1 at least and n-1 offline nodes at most, in order to keep the whole metadata available, the lowest availability (denoted as X) of metadata has to be maintained as formula (2).

$$X = \frac{C_n^2 + 1}{n + C_n^2} = 1 - 2\frac{1 - \frac{1}{n}}{n+1} \qquad n = 2,3,4,... \quad (2)$$

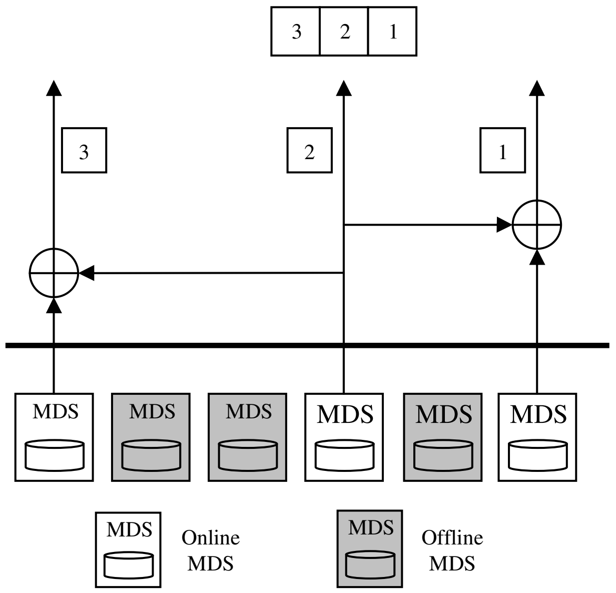

Figure 6: Metadata reading (Metadata stored in 6 MDS nodes originally, but only 3 online currently)

When storing nodes number is N and online nodes number is M, if M>=X*N, metadata can be accessed completely. Figure 6 shows the process of reading a metadata from the 3 online nodes, which is originally stored in the 6 nodes.

IV. Experiments

This paper implements a trace-driven test tools to evaluate the proposed collaborative fault-tolerance mechanism on our OSD prototype connected by a 1Gbit/sec Ethernet. System parameters for performance evaluation are listed in the table 1. Efficiency of Co-MDS has been tested in two aspects: fault-tolerance and performance.

TABLE I
SPECIFICATION OF MDS CONFIGURATION

CPU	1GHZ Pentium III
Memory	2 GB
Disk	Seagate IDE, 120GB, 7200 RPM
Network Interface	1Gbits/sec Ethernet
OS	RedHat 9, Linux kernel 2.6.12

A. *Fault-Tolerance Test*

This paper has tested fault-tolerance performance of Co-MDS under different offline nodes in a 10 nodes MDS cluster. While the number of nodes does not influence test, it somewhat reflects the scale of the network. Test policy is as follows:

1) Writes a metadata to Co-MDS when 10 nodes are online;
2) Reads the metadata while a certain number of nodes offline and repeat 100 times, then record the average availability;
3) Changes the number of offline nodes, repeating step 2

The test result is plotted in figure 7.

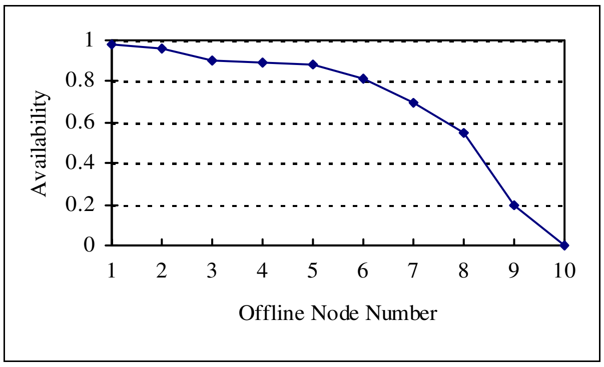

Figure 7: Co-MDS Fault-tolerance

From the figure 7, we can see that when the number of offline nodes is less than 7, Co-MDS keeps high availability; when the number of offline nodes is more than 7, metadata availability becomes low rapidly. Totally, Co-MDS can obtain good fault-tolerance under metadata server failure.

B. *Performance Test-*

This paper has also tested the performance of Co-MDS under different scale of cluster. We have conducted read, write, and mkdir operations on Co-MDS under different number of MDSes 100 times and calculated average latency. The result is shown in figure 8.

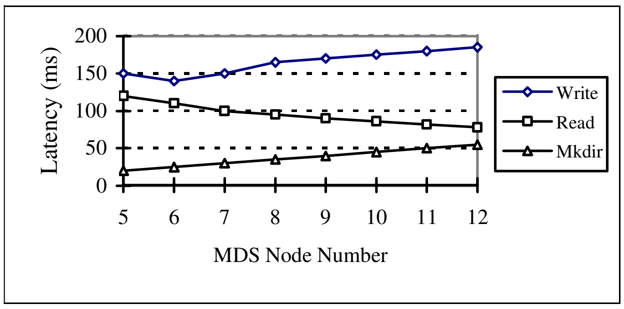

Figure 8: Co-MDS performance V.S. MDS node number

From the figure 8, we can see that global information synchronization overload increases slowly as collaborative MDS nodes manifold, which means that Co-MDS perform well in terms of performance and scalability.

V. Conclusion

This paper presents a novel distributed cooperative fault-tolerance storage mechanism for MDS, called Co-MDS. The Co-MDS consolidates disk space of individual MDS into a single storage space and provides high fault-tolerance by distributing metadata and its parity data across collaborative MDS cluster. With this mechanism, metadata is still available even when a part of metadata servers fail.

In the future work, we will modify fault-tolerance algorithm to meet increasing MDS nodes on a single metadata server cluster because our benchmarking result shows that our current solution only supports up to 7 metadata server nodes failure at the same time. Moreover, we also plan to improve the performance for a quantity of small I/O WRTITE/READ in a distributed environment.

ACKNOWLEDGE

The authors would like to acknowledge the support under the project "Design, Implementation and Management of a High Performance Object-Based Storage and Retrieval System for Large Scale Multimedia Applications", grant R-263-000-345-305 by A*STAR (Agency for Science, Technology and Research), Singapore.

REFERENCES

[1] M.Mesnier, G.R Ganger, and E. Riedel. Object-based storage. IEEE Communications Magazine, Vol. 41:pp. 84-90, 2003

[2] Thomas M.Ruwart, OSD: A Tutorial on Object Storage Devices, Proceedings of the 19th IEEE/10th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST2002)

[3] Alain Azagury, Vladimir Dreizin and Michal Factor, Towards an Object Store, Proceedings of the 20th IEEE/11th NASA Godard Conference on Mass Storage Systems and Technologies (MSST 2003)

[4] Object-based storage: The wave of storage technology and devices. Intel White Paper

[5] Information technology-SCSI Object-Based Storage Device Commands (OSD), Oct 2004, http://www.t10.org/ftp/t10/drafts/osd2/osd2r00.pdf.

[6] James S.Plank, A tutorial for Fault-Tolerate in RAID-like systems, Technical Report UT-CS-96-332 Department of Computer Science University of Tennessee July 19, 1996, 10-22

[7] X.Mplero, F.Silla, V.Santonia, Performance analysis of storage area networks using high-speed LAN interconnects, Proceedings of the 8th ISPAN Conference IEEE Computer Society. September 2000, 5-9

[8] Scott Atchley, Stephen Soltesz, James S. Plank Micah Beck and Terry Moore, Fault-Tolerance in the Network Storage Stack, Proceedings of the FAST 2002 Conference, Monterey, California, USA, January 28-30, 2002, 3-5

[9] Summeet Sobti, NitinGarg, Chi Zhang, Xiang Yu, Arvind Krishnamurthy and Randolph Y.Wang, PersonalRAID: Mobile Storage for distributed and disconnected computers, Proceedings of the FAST 2002 Conference, Monterey, California, USA, January 28-30, 2002, 5-8

[10] G.Gibson, Cost –effective high-bandwidth stoarage architecture, proceedings of ACM ASPLOS, October 1998, 4-5

[11] Xuebin He, Praveen Beedanagari, Dan Zhou "Performance evaluation of distributed iSCSI RAID", Proceedings of the international workshop on Storage network architecture and parallel I/Os, New Orleans, Louisiana