2015 44th International Conference on Parallel Processing

MAMS: A Highly Reliable Policy for Metadata Service

Jiang Zhou*†, Yong Chen*, Weiping Wang†, Dan Meng†
*Texas Tech University, Lubbock, TX, USA
†Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{jiang.zhou, yong.chen}@ttu.edu, {wangweiping, mengdan}@iie.ac.cn

***Abstract*—Most mass data processing applications nowadays often need long, continuous, and uninterrupted data access. Parallel/distributed file systems often use multiple metadata servers to manage the global namespace and provide a reliability guarantee. With the rapid increase of data amount and system scale, the probability of hardware or software failures keeps increasing, which easily leads to multiple points of failures. Metadata service reliability has become a crucial issue as it affects file and directory operations in the event of failures. Existing reliable metadata management mechanisms can provide fault tolerance but have disadvantages in system availability, state consistence, and performance overhead. This paper introduces a new highly reliable policy called MAMS (multiple actives multiple standbys) to ensure multiple metadata service reliability in file systems. Different from traditional strategies, the MAMS divides metadata servers into different replica groups and maintains more than one standby node for failover in each group. Combining the global view with distributed protocols, the MAMS achieves an automatic state transition and service takeover. We have implemented the MAMS policy in a prototyping file system and conducted extensive tests to validate and evaluate it. The experimental results confirm that the MAMS policy can achieve a faster transparent fault tolerance in different error scenarios with less influence on metadata operations. Compared with typical designs in Hadoop Avatar, Hadoop HA, and Boom-FS file systems, the mean time to recovery (MTTR) with the MAMS was reduced by 80.23%, 65.46% and 28.13%, respectively.**

***Keywords*-Parallel file systems; cluster file systems; multiple metadata service; metadata management; fault tolerance**

I. Introduction

With the arrival of the big data era, mass data processing has been widely used for extracting useful knowledge from a large amount of datasets. Most applications, including offline and online data analysis, often need long, continuous, and uninterrupted data access. Mass data storage has become a key technology for large dataset processing. Parallel/distributed file systems have attracted intensive attention in recent years and are being actively investigated to meet new demands for the management of massive datasets and a variety of data structures.

The metadata management is a critical component of parallel/distributed file systems and plays a key role in terms of their scalability, reliability, and availability. There are two metadata management mechanisms in general: a centralized mechanism or a decentralized mechanism. The centralized metadata management, including Lustre [1], Calypso [2], PVFS [3], GFS [4] and HDFS [5], has one metadata server to organize all metadata of files and directories. It greatly simplifies the design and implementation of parallel/distributed file systems but easily leads to a single point of failure: once the metadata server crashes, the file system will not be available. The decentralized metadata management, including Ceph [6], Storage Tank [7], Ursa Minor [8] and HDFS Federation [9], uses a group of metadata servers to manage the global namespace. It improves file system scalability and is the trend for future parallel/distributed file systems. However, a decentralized metadata management mechanism presents challenges in metadata operation performance and in the case of multiple metadata server failures. In this research, we focus on these challenges and introduce a highly reliable metadata management policy. We present its design, implementation, and evaluation results in this paper.

The reliability of file system metadata management has never been so important. Advances in large-scale cluster and high performance computing systems enable effective and rapid mass data processing. However, larger systems generally have more processing components and other elements, which increase the overall system failure rate [10]. Though the mean time between failures (MTBF) for an individual component may be high, the system reliability can decrease in a large scale system. As illustrated in Figure 1, the MTBF of IBM Blue Gene/L which is built with 131,000 processors is estimated to be below 7 hours [11]. In fact, recent studies indicate that servers tend to crash with 2-4% failure rate, 1-5% of disk drives die, and 2% memory errors occur per year [12]. In an extreme-scale system with more than hundreds of thousands of nodes, the MTBF is expected to fall below tens of minutes, and a node error occurs each day on average [13].

In addition to increasing hardware failures, many systems often encounter software upgrades, configuration changes, installation of new components, and planned or unplanned downtime [14], [15]. As a typical distributed framework, most Hadoop clusters will perform major upgrades every quarter and relevant software stack patch at any time. A metadata service failure can have a significant impact. For example, it will lead to tens of thousands of file system request failures at the Facebook realtime system [16] if the metadata server suspends. Due to the increasing likelihood of hardware or software failures, the metadata service can be

0190-3918/15 $31.00 © 2015 IEEE
DOI 10.1109/ICPP.2015.82

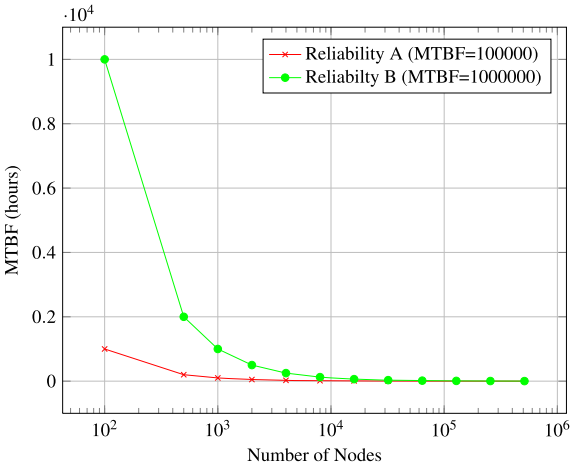


Figure 1: Reliability levels of two systems as a function of the number of nodes with MTBF of 10^5 and 10^6 hours for each node[11]

interrupted and lead to the entire file system unavailability, which has a catastrophic impact on big data applications.

To provide a highly reliable metadata service, a primary-backup strategy is often used in distributed systems such as in Lustre [1] and GFS [4]. If the primary server crashes, a backup one will take over its role and continue to respond to clients. However, there is a service downtime in the backup before it performs metadata recovery. For example, the largest HDFS cluster in Facebook with 150 million files takes about 20 minutes for failover [16]. To achieve seamless service switching, some strategies [9], [17], [18] adopt hot standby for the primary. It reduces the recovery time, but still needs dozens of seconds and minutes for recovery and still has the problem of metadata state inconsistency. Server state replication [19], [20], [21] is another way to improve metadata reliability. All metadata modifications are replicated in multiple standbys when an active is running. One of the standbys will provide the active service in case of failures. It enhances the file system reliability. Current strategies, however, suffer two limitations. First, they have influence and affect the performance of normal metadata operations. Second, they require additional failover time for state transition.

In contrast to these existing systems, we propose a novel highly reliable metadata service called *MAMS*, namely *multiple actives multiple standbys*, for failover in parallel/distributed file system. The contribution of the MAMS policy is two-fold. First, it tolerates multiple points of failures by dividing metadata servers into different replica groups and maintaining more than one standby node for recovery in each group. If the active metadata server crashes, a new active server will be elected from other standbys to take over. It ensures an automatic fault tolerance in different error scenarios, which significantly improves the file system reliability. Second, the MAMS policy reduces the overhead of recovery with little influence on file system performance. By using a prepared and interactive state transition among servers, it performs service switching in the form of hot standby. It also supports dynamically adding backup nodes at runtime. Performance results show that the MAMS policy achieved fast failover within milliseconds while achieving high performance for metadata operations.

The rest of this paper is organized as follows. Section II discusses related work. Section III describes the design of the proposed MAMS policy. Section IV evaluates the performance and characteristics of the MAMS policy, with comparisons to widely-used highly reliable metadata management strategies. Section V summarizes this research and outlines further possible work.

II. Related Work

Numerous active studies have been conducted in recent years to advance metadata service reliability for parallel/distributed file systems. We discuss existing work in this section and compare them with our work.

Traditional Primary/Backup Strategy. To support continuous service for metadata operations, the primary/backup strategy is currently widely adopted, such as in PVFS [3], Lustre [1], HARP [22], GFS [4] and HDFS [5]. It achieves fault tolerance by using the backup server to take over as the primary if the latter crashes. When the active server provides service, the backup server receives metadata modifications from it and replays journals in memory for state recovery. It has little extra overhead on normal operations but easily leads to incorrect states between the primary and backup without consistency guarantee. Furthermore, as it requires the procedure of restarting and reconnection in the backup server, it takes a long time for failover. On contrast, our proposed MAMS policy provides an automatic hot standby which significantly reduces the overhead for recovery.

Hot Standby Strategy. The hot standby strategy provides seamless service switching for the primary metadata server. AvatarNode [16] at Facebook is designed for a realtime file system that supports online applications and requirements. It uses NFS (SUN Network Filesystem) [23] to share and synchronize metadata operations among servers. In order to construct file locations, the datanodes talk to both the active and standby metadata servers. As the standby server keeps the same state with the active server, it can take over as it quickly. Hadoop HA [9] employs the Quorum Journal Manager (QJM) to share edit logs between the active and standby. They can achieve automatic fault tolerance but still take some time for recovery and has influence on normal metadata performance. Wang et al. [18] have proposed a primary-slaves topology to enable Hadoop high availability through metadata replication. The mechanism consists of one primary critical node and several slave nodes

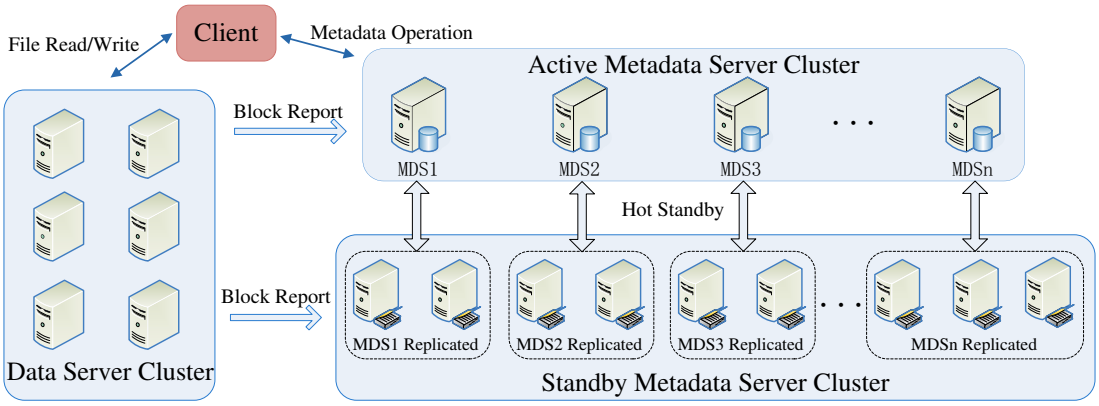


Figure 2: Highly reliable multiple actives multiple standbys (MAMS) metadata service for file systems

for removing a single point of failure. It spends about three times of the response delay in metadata operations. Different from the strategies discussed above, our proposed MAMS policy provides fault tolerance for multiple metadata servers in the file system. MAMS divides metadata servers into different replica groups and maintains more than one hot standby server for fail-over in each group. It significantly improves the file system reliability with less influence on metadata operations.

Server State Replication Strategy. To further enhance system reliability, some strategies adopt replicated state transition for designing highly reliable services. TidyFS [19] maintains multiple metadata servers as a replicated component for the centralized metadata server. It leverages the Autopilot Replicated State Library [24] to replicate the metadata and operations on that metadata using the Paxos [25] algorithm (protocols for solving consensus in a network of unreliable processors). Boom-FS [20] implements a data-centric analytics stack which is compatible with Hadoop interfaces. To achieve reliability, it adopts a globally-consistent distributed log to guarantee a total ordering over events affecting replicated states. Both of them use the Paxos algorithm to maintain consistency across servers and provide a reliable mechanism by the replicated state machine approach. The operation performance, however, is affected for centralizing repair action decisions and state transition, which leads to additional failover time. Our proposed MAMS policy addresses this issue.

Active/Active Model Strategy. Symmetric or asymmetric active/active models are also used to improve file system reliability. In the symmetric strategy [26], more than one dedicated server provides the shared global state with the virtual synchrony technique. With a fast delivery protocol, service node failures do not cause a fail-over to a backup and there is no disruption of service or loss of service state. Asymmetric strategy [27] comprises $n+1$ and $n+m$ configurations with n active nodes and 1 or m standby nodes. The standby servers monitor all active servers and perform failover when the outage is detected. As client requests are distributed to multiple active servers, these strategies will result in a performance decrease in metadata processing and additional overhead for response time. In contrast, our proposed MAMS policy adopts distributed coordination and automatic state transition to ensure failover in different error scenarios. It achieves a faster transparent fault tolerance within milliseconds while delivering high performance for metadata operations.

III. MAMS Policy

To achieve a highly reliable metadata service in a parallel/distributed file system, we propose a new MAMS (multiple actives multiple standbys) policy that uses an active-standby cluster for fault tolerance. By maintaining a prepared and automatic state transition, the MAMS policy achieves quick recovery in the form of hot standby. We introduce the detailed design and implementation in this section.

A. System Model

In the proposed highly reliable MAMS metadata service, multiple metadata servers are deployed to manage a global namespace for file systems. Hash-based methods [28] are adopted for namespace partitioning and metadata distribution. Dedicated backup nodes are deployed to improve the reliability of the metadata server cluster. The backup node can have two states, a standby state and a junior state, discussed in detail below. Figure 2 shows the architecture of the highly reliable MAMS metadata service. For each metadata server (MDS), a replica group is constructed with one active and two or more standby nodes. By state replication, the standby nodes can keep the same states with the active and take over its role in the event of failures. As the distributed metadata service decouples data and metadata management,

file contents are split into blocks and replicated in the data server cluster. Block locations are periodically reported to both the active and standby nodes by data servers. It means that the standby node has the up-to-date file locations and can achieve a hot standby for the active server.

In the replica group, only one metadata server is regarded as an active and provides service. Other backup nodes keep their namespace state consistent with the active. An SSP (shared storage pool) [28] is designed for metadata synchronization. The pool is built on existing active or backup servers and needs no additional device or third-party software support. When the active stores metadata modifications or namespace image, it writes them sequentially as shared files in the SSP. Through a modified two-phase commit protocol [29], the standby achieves journal synchronization from the active with this pool.

For each replica group in the MAMS, the metadata server starts up in different states according to a global view. The metadata server acts as one role and is converted to other states at different conditions. The state of server is divided into three categories as belows:

- Active. An active server receives client requests and provides metadata service for the namespace partition it manages. At any replica group, exactly only one of servers is in an active state at any time.
- Standby. A standby server is a backup node which simply keeps an up-to-date namespace state with the active at any time. It does not provide metadata service but can take over as the active server in the event of failures.
- Junior. A junior server is a type of backup node but cannot provide hot standby. It is in an intermediate status in which the state does not synchronize with the active server. It can be a server which restarts after a failure, or is a newly added backup node.

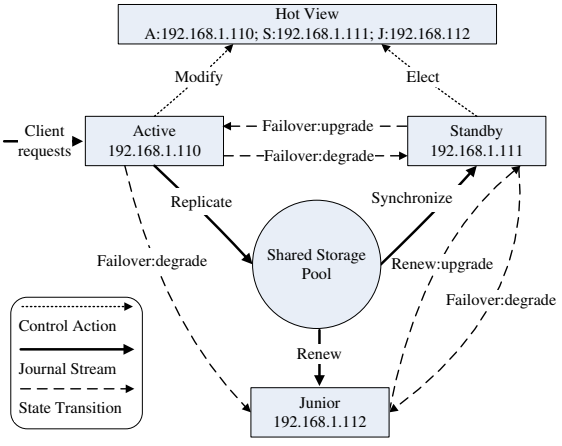


Figure 3: Server state transition in the replica group

Under different conditions, the state of servers may be switched to each other. Figure 3 shows a diagram of server state transition in the replica group. It can be seen that there are three servers in different states: one is an active server with the IP address 192.168.1.110, one is a standby server with the IP address 192.168.1.111, and another one is a junior server with the IP address 192.168.1.112. Two distributed protocols are responsible for the state transition. The failover protocol performs upgrading or degrading between the active and standby servers. It also degrades them to the junior state when necessary. The renewing protocol is adopted to upgrade the junior to a standby. Based on the SSP, the active server synchronizes and replicates journals to standby for keeping the state consistent. It reduces additional overhead with little influence on normal metadata operations. A monotonically increasing serial number (sn) is assigned by the active when it writes journals. Each batch of log records is described with the pair $<sn, transactionid>$. By subtracting the values between two sn, the junior can retrieve missing metadata from the SSP or the active server for upgrading to the standby state.

B. Active Election

As the MAMS policy uses more than one standbys for fault tolerance, it needs to elect a new one when the active server crashes. With the Paxos algorithm for consensus [25], MAMS ensures that only one active is elected each time. The process of active election is like distributed lock management. When the active crashes, each standby tries to obtain a distributed lock periodically until it either succeeds or encounters a failure. Active election can also be achieved by comparing log sn values. It ensures the continuity of metadata service even if no standbys are in the global view. Algorithm 1 describes the active election algorithm in a replica group.

Algorithm 1 Active election in the replica group

```
1:  stdby = the number of standby nodes
2:  if active has errors or no active in the global view then
3:    if stdby > 0 then
4:      while no active is elected do
5:        each standby generates a random number
6:        the standby S_i which has the largest random
          number obtains the lock and is upgraded to active
7:      end while
8:    else
9:      while no active is elected do
10:       selecting the junior J_i with maximum sn in logs
11:       J_i obtains the lock and takes over as the active
12:     end while
13:   end if
14: end if
```

C. Active-Standby State Transition

The MAMS achieves an automatic active-standby switch for fault tolerance. Combined with the heart beat from the node monitor, the MAMS uses an event-driven mechanism

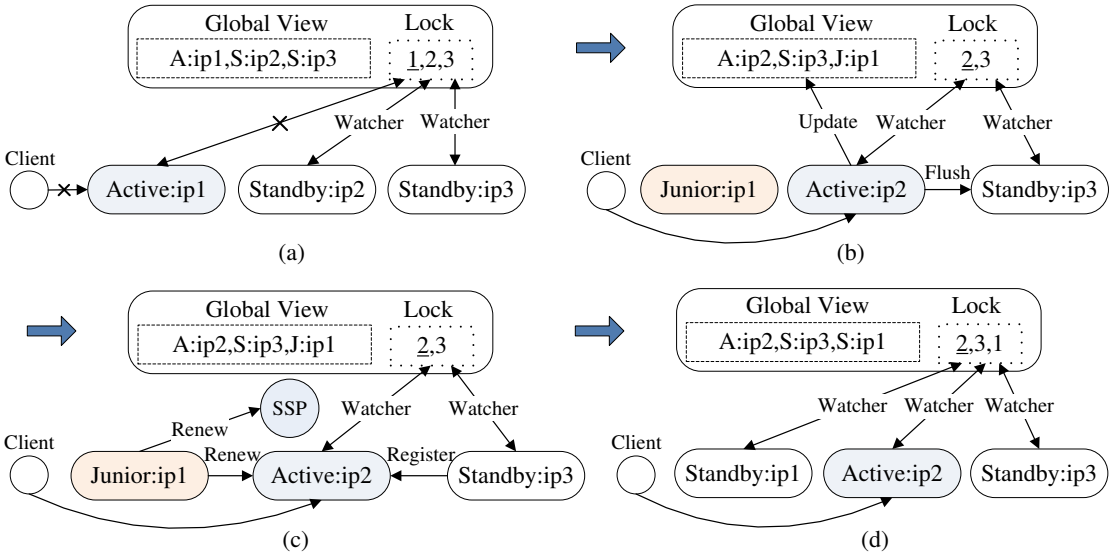


Figure 4: The main procedure of active-standby transition

to trigger active election and state transition in case of failures. Each server has three event watchers on the global view: one is on itself, one is on the active server and another is on the distributed lock. Any error will trigger them and results in two situations: the active state is changed, which makes the active server loses the lock and an election process is launched, or other state transition between standby and junior. It ensures that no processes can obtain the distributed lock before the active loses it. The main procedure of active-standby transition for failover protocol is depicted in Figure 4, in which the underlined number means the server with the same IP sequence that has granted the lock.

When the active server has detected failures, it stops providing service and no longer responds to clients, as shown in Figure 4(a). In this example, the active is directly degraded to junior. But some obsolete data, such as buffered journals in the active server, may be flushed to standbys and shared log files in the SSP. It does not matter because the standby only receives and responds for journals which come from the active server. As the global view will be modified immediately and the event is triggered whenever there are changes in server states. Thus, there is no scenario that two metadata servers access the same shared file simultaneously in which it achieves the function of IO fencing.

Once a standby server obtains lock successfully, it holds the lock and prepares for state transition. Events are triggered to notify others to stop competing which will reduce unnecessary actions for election. The elected standby then does not receive any journals from the active and awaits an opportunity to switch. If there are no pending or processing operations, it will launch upgrade immediately. Otherwise, the elected standby applies them to its own namespace and ignores all new modifications. After committing cached journals, it enters an upgrade procedure which is shown in Figure 4(b) and (c):

1) The elected standby visits the global view and checks its own state. If it is in a junior state, it must stop upgrading and give up the lock. The re-election action will be performed at this time.
2) The elected standby modifies relevant states in the global view. It changes the state of previous active to standby or junior and sets itself to active. At this moment, operations from the previous active will be refused by all nodes.
3) New requests from clients and read service are allowed. Once server states are switched, new file operations may reach the elected standby. It receives and saves requests in memory but does not commit them until the upgrade process is finished.
4) To avoid missing operations (e.g., the previous active does not return success to all clients), the elected standby flushes last cached journals to others in the replica group again. As the previous active may become standby and receive the same journals again, duplicated journals must be distinguished in this step. Each standby will decide whether to commit logs by comparing values of sn. Only if sn from the active is larger than the current maximum serial number, the standby applies journals and responds to it.
5) The elected standby receives register information from all servers in the replica group. It confirms and changes the state of each server in the global view. If a server does not have the same maximum sn, it is switched to junior. Otherwise the server will be assigned to standby

or junior according to its previous state.

6) If more than one standby is registered successfully, the elected standby becomes the new active. It will launch the renewing process for junior at appropriate time.

When a junior is upgraded to the standby state, the global view turns back to a stable status which is shown in Figure 4(d). During the process of state transition, the elected standby will stop upgrading if any failures occur. The MAMS will launch the re-election process in the replica group at this time. Benefiting from our namespace partition strategy [28], the client can reconnect to the new active directly and automatically after active-standby switching and resend requests when needed. As the process is completely transparent to applications, the file system sees no errors occur in the case of failures.

D. Junior Renewing

Once the active has detected fatal errors, such as disk failures, it will be directly degraded to the junior state. The standby is often converted to the junior state when errors occur. The junior is an intermediate state which cannot provide a hot standby. As the active does not synchronize log records to the junior, there is inconsistence in namespace states between them. With the reduction of standbys, the file system will turn into an unreliable status. To avoid this risk, the MAMS adopts the renewing protocol to make the junior recover missing operations and become the standby.

During the runtime, the active scans the global view periodically and tries to launch the renewing process when there are juniors. It selects one server with the least gap in namespace state and creates a session for recovery at each time. In response, the junior receives commands and starts the task for upgrading. The active decides the renewing strategy according to the value of maximum sn from the junior. If the difference between them is large, the junior first retrieves the namespace image or journal files. Otherwise, it synchronizes metadata modifications from the active directly. As metadata files are stored in the SSP, the junior may obtain them locally from the pool and reduce the transmission latency. During the course of reading images, the junior reconstructs the tree of files and directories in memory and reaches a consistent state with the active gradually. As there is no sn associated with it (default 0) and this phase can spend a long time, the junior records the checkpoint that has been committed. It can continue to recover from other replicas in the last position and avoid retransmitting the whole files if there are any interrupts in the process.

After loading the namespace image, the junior asks a list of journals from the active which can decide missing files according to the image timestamp. The junior then starts to apply logs from the active or the SSP. All above operations are performed in the background which does not affect active service. During the procedure, the junior records the current sn and sends it to the active periodically. When there are few gaps in the values of sn, the active launches final synchronization stage. Once the junior recovers all journals and returns the same sn, the active modifies the global view and changes its state to standby. Then the junior is upgraded and can receive metadata modifications from the active. It keeps an up-to-date namespace state in the form of hot standby. By renewing, more new backup nodes can also be added in the replica group at runtime. It significantly improves the reliability and availability of metadata service.

IV. Evaluation

In this section we present the evaluation results of the proposed highly reliable metadata management policy MAMS. A prototyping file system CFS (Clover File System)[28] has been designed and implemented with the MAMS policy for the validation and evaluation. It adopts multiple metadata servers to manage a global namespace and hash-based method for metadata partition. The CFS is based on the SSP for metadata synchronization and uses the MAMS policy for reliable metadata management.

The experimental platform is a cluster with 20 nodes. Each node consists of four Intel Xeon X3320 Processors, 8-GB memory and one Gigabit network interface card, with Linux kernel 2.6.32. Each of them acts as the pool node in the SSP and stores image and journal files. For file system operations, multiple metadata modifications are aggregated before being submitted and written back to journals in an asynchronous way. To facilitate failure detection and auto service switch, the Zookeeper was used to monitor nodes, trigger events and maintain the consistent global view.

A. Overhead on Metadata Operations

The MAMS policy provides a highly reliable mechanism for metadata service, though it may have an impact on normal metadata operations. To measure the overhead on normal metadata operations, the experiments were conducted under failure free cases. As the CFS is an implementation of multiple metadata servers based on the HDFS, we first compare it with the HDFS by configuring different active and standby nodes. Then we compare the MAMS with typical highly reliable systems, including BackupNode [5], Hadoop AvatarNode [16], and Hadoop HA [9], to observe the metadata operation performance.

Figure 5 shows the performance of HDFS and CFS with the MAMS policy. The CFS was configured with three metadata servers in which each active had 1 to 4 standbys, e.g. MAMS-3A3S means 3 actives and 3 standbys. The HDFS adopts a single metadata server without any reliable mechanism. The tests used multiple clients on different nodes to provide the workload with each performing 1 million operations.

From Figure 5, it can be observed that the performance of CFS is higher than the HDFS for *create* and *getfileinfo* operations. This is due to the metadata partition strategy

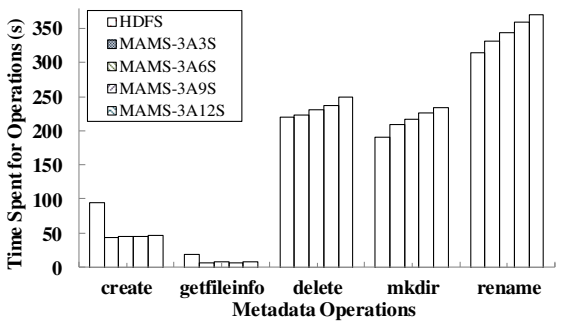


Figure 5: Performance of MAMS with different active and standby nodes

in CFS which support performing the operations simultaneously. The other three types of operations, including *delete*, *mkdir* and *rename*, belong to distributed transactions in the CFS. The state synchronization among servers will have an impact on these operations. For each of these operations, except the read-only operation *getfileinfo*, the performance of the CFS was gradually decreased with adding standbys in each replica group. As the active needs to synchronize journals to more nodes when more standbys were added, it increased the synchronization time. The MAMS policy, however, was built upon the SSP which reduced the synchronization overhead and was not a performance bottleneck. The additional overhead is negligible and there was a minor performance degradation. For example, the performance of the *rename* operation with 3A3S was declined with 3.89%, 4.28% and 3.25% respectively by adding one standby for each metadata server. The MAMS policy decreases some operation performance of the CFS but significantly enhances the metadata service reliability (in general, any reliable metadata management strategy sacrifices the performance for reliability). It is worthy to trade slight performance reduction for the entire file system reliability in return.

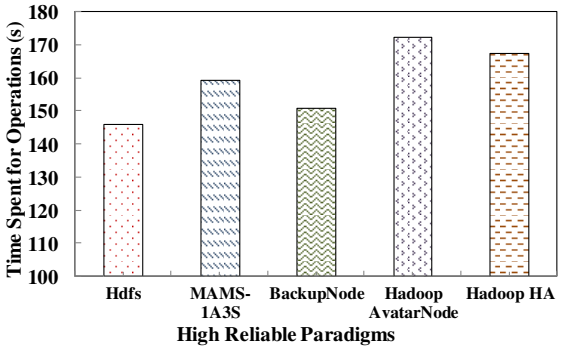


Figure 6: Comparison on metadata operation performance with different reliability mechanisms

Figure 6 shows the comparison on metadata operation performance with different reliable metadata management mechanisms. Each test was performed with 1 million operations with mixed *create*, *getfileinfo*, and *mkdir* operations. The CFS was configured with 1A3S and other three systems adopt the primary-backup strategy. The experiment results reveal that the metadata operation performance was reduced with reliable metadata management mechanism compared to the HDFS. It is because that all these there systems need real-time state synchronization between the primary and backups, which results in additional overhead. The BackupNode incurred less time but it does not guarantee metadata consistency. For the CFS, the performance was higher than the Hadoop AvatarNode and Hadoop HA even using multiple standbys. This is mainly due to two aspects: one is the journal synchronization strategy with the SSP which greatly reduced the additional overhead; the other is the MAMS policy can effectively perform service switching and achieve failover.

In summary, the experiment results indicate that the MAMS policy can significantly improve metadata service reliability and keep server states consistent in the same replica group with little effect on the performance.

B. Failover Time

This series of tests measured the failover time of different reliable metadata management systems discussed in the above subsection. They are all implemented based on the HDFS and increase the metadata reliability guarantee compared to the vanilla HDFS. We also observed and analyzed the time needed for different stages in the MAMS in these tests, which includes active election, active-standby switching and client reconnection. The CFS was configured with 1A3S and other systems adopt the primary-backup strategy. In the Hadoop HA, the number of JournalNodes was set to 4. For fault detection, the heart beat interval and session timeout of ZooKeeper were set to 2 and 5 seconds, respectively. Metadata server failures were generated by shutting down processes, unplugging the network cables, etc.

During the experiments, client interfaces were called to access the file system continually and the failover time was measured. We choose the mean time to recovery (MTTR) as the metric for comparison. It can be computed by subtracting the timestamp of operation failure and that of success, when metadata service is unavailable and recovered, respectively. Each test was performed 10 times, and the average MTTR was computed as below:

$$MTTR = \frac{\sum_{k=1}^{times} (Time_{return\ failure} - Time_{return\ success})}{Times}$$

Table I reports the MTTR of different systems. The first column lists the image size which indicates file system scale. The tests were conducted using *create* operations to create files. There are more than 7 million files when

Table I: MTTR of different reliable metadata management systems

Image (MB)	MTTR (s)			
	MAMS-1A3S	BackupNode	Hadoop Avatar	Hadoop HA
16	5.893	2.784	27.362	15.351
32	6.376	5.326	31.574	17.439
64	6.531	9.653	30.721	18.624
128	5.742	22.928	29.273	16.372
256	5.436	36.431	32.805	19.016
512	6.795	78.365	31.446	17.853
1024	6.081	142.513	33.239	19.193

the image size is about 1GB. We can observe that the MTTR of BackupNode increased gradually with file system scale expansion from 2.784 to 142.513 seconds. This is mainly because its backup node needs to recollect block locations before taking the place of the primary. Compared with the BackupNode, Hadoop Avatar, and Hadoop HA, the average failover time of the MAMS are 14.35%, 19.77%, and 34.54% of them respectively. The results verified two aspects. One is the process of failover in the MAMS was performed with a relatively low overhead. The other is that clients were able to automatically connect to the new active rapidly and achieved transparent fault tolerance.

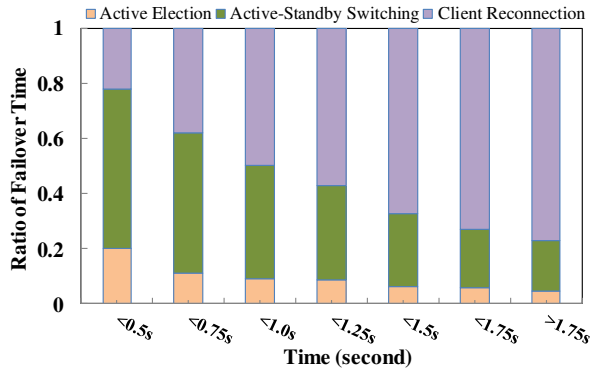


Figure 7: The proportion of failover time at each stage in MAMS

Table I shows the benefits of the MAMS for recovery. To further analyze the overhead in MAMS, we counted the proportion of failover time at each stage. Figure 7 reports the results with the same analysis like in Table I. The horizontal axis represents the failover time distinguished by values exclusive of session timeout (default 5 seconds). From Figure 7, we can see that it spent the least time for active election during recovery (less than 100ms). This is mainly because the MAMS uses the event trigger mechanism to deal with failures and the Paxos algorithm for election. It can reach a consensus among the active and standbys quickly. Meanwhile, the time of active-standby switching is also stable with values between 250ms and 350ms. With the growth of failover time, the proportion of client reconnection was increased. It indicates that our MAMS policy not only reduced failover time to milliseconds but also improved file system reliability compared to other strategies.

C. *Efficiency for Recovery*

In the above subsection, we have reported the failover time comparing the MAMS with other reliable metadata management systems. These tests were conducted based on a single point of server failure. This subsection verifies the efficiency of the MAMS for recovery when multiple server errors occur. The CFS was configured with one replica group including one active and three standbys (1A3S). We used multiple client processes on different nodes to provide overload. The tests include continuous *create* and regular *mkdir* operations. Files are distributed among multiple directories with average requests per second being counted. Table II reports different test scenarios and corresponding server state transition. The states of nodes include active (denoted as A), standby (denoted as S) and junior (denoted as J). The symbol "-" means the server is still in a fault state.

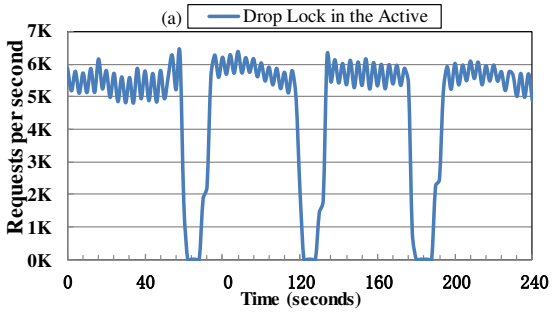
Three groups of tests were performed in our experiments. Errors were generated through modifying the global view to make the active lose the lock (Test A), unplugging and reconnecting network wires (Test B), and restarting processes (Test C). Figure 8 shows the failover ability of metadata operations in the CFS. The vertical axis is average requests per second and the horizontal axis is the test time.

As shown in Figure 8(a), the active lost the lock at the moment of 60 seconds. It triggered an active election and state transition in response to state 2 in Table II. At this point, the average time of metadata operations was not reduced to 0 immediately. It is because the active has returned partial success results to clients. Subsequently, the active stopped the service and performed a fault-tolerant processing from 62 to 68 seconds. The file system did not respond to clients anymore. After service switching, the state of each node was transformed to state 4. The original active registered to the new one as a standby. When the new active began to provide service, clients may resend requests for incorrect results. Therefore, the curve has a slight upward trend at the time of 70 seconds. The duplicated message handling in the MAMS will avoid the problem of incorrect metadata operations. At last, the file system restored to the normal performance before failures in which the new active continued to provide metadata service. The same operations were also performed at the time of 120 and 180 seconds. Experiment results indicate that the MAMS policy can elect the new active and achieve server takeover quickly to keep the continuity of file system service.

Test B generated network errors by taking out/plugging back wires and the results are shown in Figure 8(b). The test made two servers failed for each time. With the fault

Table II: Test scenarios and server state transition

State	Active lose lock (Test A)				Take out/plug back network wires (Test B)				Shut down and restart processes (Test C)			
	MDS	BN	BN	BN	MDS	BN	BN	BN	MDS	BN	BN	BN
1	A	S	S	S	A	S	S	S	A	S	S	S
2	—	S	S	S	A	S	—	—	—	S	S	S
3	—	A	S	S	A	S	J	J	—	A	S	S
4	S	A	S	S	A	S	S	S	J	A	S	S
5	S	—	S	S	—	—	S	S	S	A	S	S
6	S	—	A	S	—	—	A	S	S	—	—	S
7	S	S	A	S	J	—	A	S	S	—	—	A
8	S	S	—	S	S	—	A	S	S	J	—	A
9	S	S	—	A	S	J	A	S	S	S	J	A
10	S	S	S	A	S	S	A	S	S	S	S	A



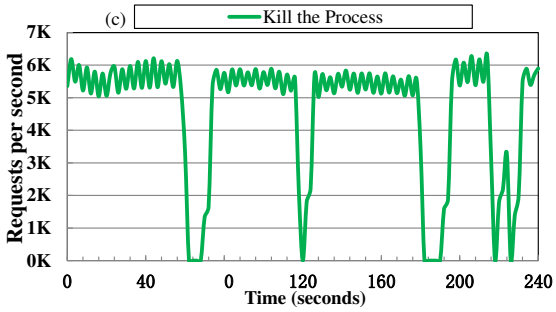


Figure 8: Failover ability of metadata operations

tolerance capability, the file system stopped responding to clients temporarily but then returned to normal functions quickly. As the CFS can continue working in the case of at least one standby, the file system still provides metadata service even when the active and standby fail simultaneously. Experiment results confirmed that the MAMS policy can tolerate multiple points of failures for metadata servers and achieved upgrade for juniors by the renewing protocol. It significantly improves the reliability of the file system. Test C generated process errors and similar results were observed, as shown in Figure 8(c). These experiment results verify the efficiency of the MAMS policy well.

D. Transparent Failover for MapReduce Programs

The CFS file system provides compatible interfaces with the HDFS and supports mass data processing in the MapReduce framework. To verify transparent failover for upper applications, we compared the CFS with the Boom-FS, another typical reliable system with multiple metadata servers based on the HDFS. The test generated a metadata server error for measuring the program completion time with the same environments. It run a Hadoop wordcount job on a 5GB input file. The CFS was configured with 3A9S.

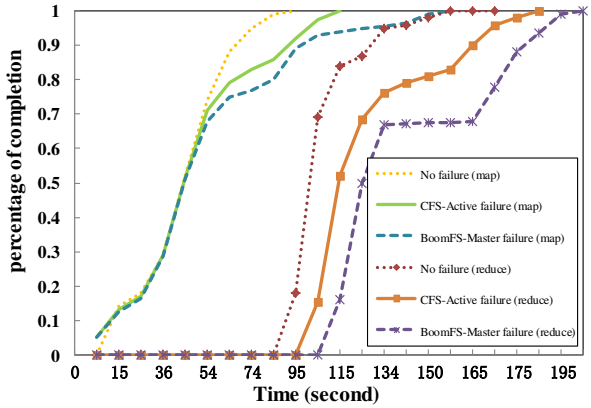


Figure 9: Run time comparison for MapReduce programs in case of failures

Figure 9 shows the cumulative distribution of the per-

centage of completed MapReduce jobs over time in case of failures. The data of Boom-FS including normal and failure operations comes from [20]. As shown in the figure, it had an influence on the program when metadata server failed. Compared with the Boom-FS, the CFS exhibited better failover performance when errors occurred. Its completion time of map and reduce jobs was less than the Boom-FS, by 28.13% and 9.76% respectively. The reduce jobs of the Boom-FS have a phenomenon of suspension. As it took time to finish map jobs for recovery, the reduce jobs needed the former to write intermediate results into the file system before continuing subsequent operations. It added additional waiting time. On the contrary, the fault tolerant mechanism in the MAMS ensured fast taking over for metadata service. It had little effect on the MapReduce execution. The experiment results indicate that our MAMS policy can achieve transparent failover for upper applications and ensured expected behavior in case of failures.

V. Conclusion

This paper introduces a novel highly reliable MAMS (multiple actives multiple standbys) metadata management policy for metadata service in parallel/distributed file systems. It supports application scenarios such as mass data analysis and processing which require continuous and uninterrupted services. The MAMS policy uses the active-standby cluster to achieve automatic recovery in case of failures. Compared with traditional reliable mechanisms, it has several notable advantages. The MAMS policy divides multiple metadata servers into different replica groups and maintains more than one backup node for each active server. It is based on a built-in shared storage pool which reduces the overhead for state synchronization. Combining a global view with two distributed protocols, the MAMS achieves an automatic state transition and service takeover in the form of hot standby. We implemented the MAMS policy in the CFS (Clover File System), a large-scale distributed file system compatible with the Hadoop platform. However, the policy can also be used in other parallel/distributed file systems for better system reliability. The experiment results show that the MAMS policy not only achieved faster failover with less influence on the performance but also significantly improved metadata service reliability.

In the future, we plan to continue improving file system reliability by exploring other namespace management methods and data recovery at any point with less data loss.

Acknowledgment

This work is supported by the National High-Tech Research and Development Program of China under grant 2013AA013204, by the Strategic Priority Research Program of the Chinese Academy of Sciences under grant XDA06030200, by the National HeGaoJi Key Project under grant 2013ZX01039-002-001-001, and in part by the National Science Foundation under grant CNS-1338078 and IIP-1362134 through the Nimboxx membership contribution.

References

[1] P. J. Braam, "The Lustre storage architecture," White Paper, Cluster File System, Inc., Oct. 2003.
[2] M. Devarakonda, B. Kish, and A. Mohindra, "Recovery in the Calypso file system," *ACM Trans. Comput. Syst.*, vol. 14, no. 3, pp. 287–310, 1996.
[3] P. H. Carns, W. B. L. III, R. Ross, and R. Thakur, "PVFS: A parallel file system for Linux clusters," in *Proc. of the 4th Annual Linux Showcase and Conference*, Atlanta, USA, Oct. 2000, pp. 391–430.
[4] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," in *Proc. of SOSP'03*, New York, USA, Oct. 2003, pp. 29–43.
[5] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. of MSST'10*, Incline Village, USA, May 2010, pp. 1–10.
[6] 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 *Proc. of OSDI*, 2006, pp. 307–320.
[7] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg, "IBM Storage Tank-a heterogeneous scalable SAN file system," *IBM Systems J.*, vol. 42, no. 2, pp. 250–267, 2003.
[8] S. Sinnamohideen, R. Sambasivan, J. Hendricks, K. Liu, and G. Ganger, "A transparently-scalable metadata service for the Ursa Minor storage system," in *USENIX Annual Technical Conference*, Boston, USA, Jun. 2010.
[9] "Hdfs federation," 2014. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/Federation.html.
[10] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Dependable Sec. Comput*, vol. 7, no. 4, pp. 337–351, 2010.
[11] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *J. of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013.
[12] D. Fiala, F. Mueller, C. Engelmann, K. Ferreira, R. Brightwell, and R. Riesen, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proc. of SC'12*, 2012.
[13] K. Sato, A. Moody, K. Mohror, T. Gamblin, B. R. D. Supinski, N. Maruyama, and S. Matsuoka, "FMI: Fault tolerant messaging interface for fast and transparent recovery," in *Parallel and Distributed Processing Symposium*, May 2014, pp. 1225–1234.
[14] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the shrew: Modeling the normal and faulty behaviour of large-scale HPC systems," in *Proc. of IPDPS'12*, May 2012, pp. 1168–1179.
[15] E. Berrocal, L. Yu, S. Wallace, M. E. Papka, and Z. Lan, "Exploring void search for fault detection on extreme scale systems," in *International Conference on Cluster Computing*, 2014, pp. 1–9.
[16] D. Borthakur, J. S. Sarma, J. Gray, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer, "Apache Hadoop goes realtime at Facebook," in *Proc. of SIGMOD'11*, Jun. 2011, pp. 1071–1080.
[17] A. Oriani and I. C. Garcia, "From backup to hot standby: High availability for HDFS," in *Proc. of Symposium on Reliable Distributed Systems*, California, USA, Oct. 2012, pp. 131–140.
[18] F. Wang, J. Qiu, J. Yang, B. Dong, X. Li, and Y. Li, "Hadoop high availability through metadata replication," in *Proc. of the first international workshop on Cloud data management (CloudDB)*, New York, USA, 2009, pp. 37–44.
[19] D. Fetterly, M. Haridasan, M. Isard, and S. Sundararaman, "Tidyfs: A simple and small distributed file system," in *USENIX Annual Technical Conference*, 2011, pp. 34–34.
[20] P. Alvaro, T. Condie, N. Conway, K. Elmeleegy, J. M. Hellerstein, and R. C. Sears, "Boom: Data-centric programming in the datacenter," Technical Report UCB/EECS-2009-113, Tech. Rep., 2009.
[21] J. Lin, F. Leu, and Y. Chen, "ReHRS: A hybrid redundant system for improving MapReduce reliability and availability," *Modeling and Processing for Next-Generation Big-Data Technologies*, vol. 4, pp. 187–209, 2015.
[22] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shrira, and M. Williams, "Replication in the Harp file system," in *Proc. of SOSP*, 1991, pp. 226–238.
[23] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the Sun network filesystem," in *Proc. of the Summer USENIX conference*, Portland, USA, Jun. 1985, pp. 119–130.
[24] M. Isard, "Autopilot: Automatic data center management," *ACM SIGOPS Operating Systems Review*, vol. 41, no. 2, pp. 60–67, 2007.
[25] L. Lamport, "The part-time parliament," Research Report 49, Tech. Rep., 1989.
[26] X. He, L. Ou, C. Engelmann, X. Chen, and S. L. Scott, "Symmetric active/active metadata service for high availability parallel file systems," *J. Parallel Distrib. Comput.*, vol. 69, no. 12, pp. 961–973, 2009.
[27] C. Leangsuksun, T. Liu, S. L. Scott, V. K. Munganuru, and C. Engelmann, "Asymmetric active-active high availability for high-end computing," in *Proc. of the 2nd International Workshop on Operating Systems*, Jun. 2005.
[28] Y. Wang, J. Zhou, C. Ma, W. Wang, D. Meng, and J. kei, "Clover: A distributed file system of expandable metadata service derived from HDFS," in *International Conference on Cluster Computing*, 2012, pp. 126–134.
[29] M. T. Ozsu and P. Valduriez, "Principles of distributed database systems," Springer Science and Business Media, Tech. Rep., 2011.