CHIRAYU: A Highly Available Metadata Server for Object Based Storage Cluster File System

Sapna Bafna sapnabafna@ieee.org
Shruti Dalvi shruti_dalvi@ieee.org
Abhinay Kampasi abhinaykampasi@ieee.org
Aditya Kulkarni adityakulkarni@ieee.org

Final Year Engineering,
Computer Science Department,
Pune Institute of Computer Technology,
Dhankavdi, Pune-411043, India.

Abstract: *The Lustre File System architecture supports Object Based Storage. It separates data and metadata path of file system operations. An outage of the Metadata Server could considerably degrade the system performance. Providing a backup Metadata Server will increase the availability of Lustre File System to 99.9%. The backup server monitors the primary server over the network and shared storage to detect failure. On failure of the primary server, the backup builds the state that was on the primary, and takes up its tasks. The first request from the clients triggers the recovery process on them. All further requests are queued till the completion of the recovery and clients thus receive non-blocking service from the server.*

Index Terms: *Object Based Storage, Lustre File System, Metadata Server, High Availability, IP takeover*

1 Introduction

Traditionally, three tier architectures consist of application, file system and storage device. These layers can be configured in various ways to obtain storage architecture flavors like Direct Attached Storage (DAS), Network Attached Storage (NAS) and Storage Area Network (SAN). These are fundamentally block-based technologies. NAS supports interoperability but projects poor storage performance. On the other hand, SAN assures high performance. The need to integrate the strengths of these three architectures into a single framework gave rise to *Object Based Storage (OBS)* [2].

Object Based Storage is a level higher than a block-level method but one level below a file-level access method. It is not intended to replace either block-level or file-level access methods but rather to add a needed layer of abstraction that sits between them and thus provides the required amount of intelligence with the storage device. This moves the storage management functionality with the storage layer itself. The unit of storage is an *object* [2]. Object Based Storage is an aggregation of logical components, which run on a cluster of systems. The three components are the Metadata Server (MDS), Object Storage Targets (OST) and the Client File System (CFS).

The Metadata Server is used as a global resource to find the location of objects. Object Storage Targets perform block allocation for data objects, leading to distributed and scalable allocation of metadata. The Client File System now becomes simply a File Manager: an abstraction layer between the user application and the OST. Requests from the CFS first go to the MDS and then there is direct data transfer between the CFS and the OST. This makes the system highly scalable.

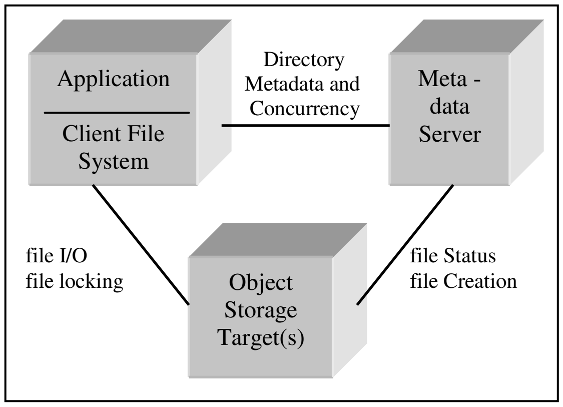

Fig. 1: Lustre File System Overview

The Lustre File System [1] architecture is based on Object Based Storage. The USA National Laboratories and Department of Defense (DOD) started exploring Lustre as a potential next generation file system [1]. The development of the system continued in aim to serve for the same. To facilitate the compute-intensive operation that was required, Lustre was required to not only yield excellent performance but also be scalable. The current implementation of Lustre supports 1000's of clients and 100's of OST's, managed by a single MDS [4]. Clearly, the Metadata Server is the most crucial part of the Lustre system and its uninterrupted and consistent working is of prime importance for efficient working of the system as it is a single point of failure.

This paper describes the design and implementation of a Highly Available Class 3 [8] Metadata Server (HA-MDS) for Lustre. We have introduced a backup Metadata Server to monitor the primary server.

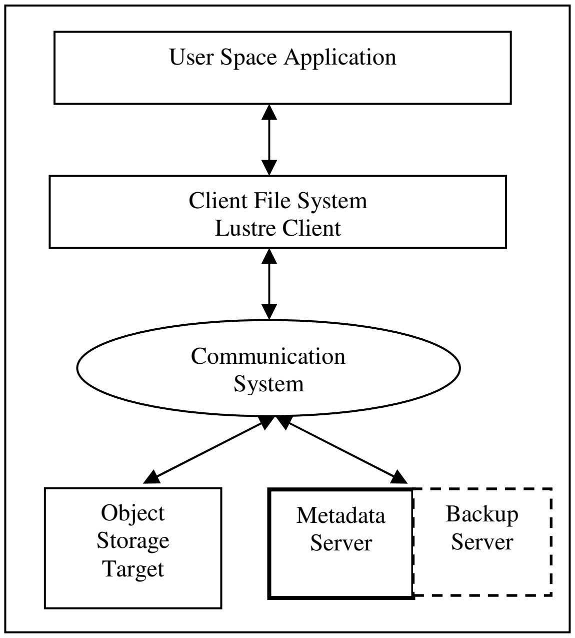

Fig. 2 System Overview

The high availability subsystem consists of mechanisms to detect failure of the primary MDS, failover the backup MDS and re-establish lost connections with the clients. In section 2, we describe the design goals of this system. In section 3, we detail the system design. Section 4 characterizes the system. In section 5, we compare our test results with the current Lustre system. Finally, we draw conclusions and outline future work in Section 6.

2 Design Goals

The primary goals of a Highly Available (HA) system are:

- Detecting failure of primary server.
- Failover.
- State transfer to backup server.
- Re-establish client connections.
- Serve pending requests.
- Continue normal request processing.

The primary MDS could suffer a network problem, its shared storage access could fail or the system itself could go down. The backup server is responsible for detecting any of these or a combination thereof. The primary MDS needs to send packets through the network to the backup MDS to ensure that it is able to access the network. Similarly, it is made to continuously write to a file which is read by the backup MDS to ascertain that it is able to access the storage as well.

In case of a system crash, both of the above tests would diagnose the failure of the primary MDS. In case of only a network or storage access failure, it is the responsibility of the Failover part of the backup server to ask the primary to bring itself down, as it can no longer serve.

The Metadata Server stores information on a persistent storage. Vital among this information is the Lustre metadata and client transaction information. At any instance, if the primary MDS fails, the backup server should be able to obtain this information essential to build the same state as that on the primary MDS. This can be achieved by maintaining a content storage shared by the two servers. The server side recovery can thus be done.

The client part of the recovery would only start when the client makes the first request during or after the secondary metadata server takes over. This starts with reestablishing the connection with the MDS, transparent to the client request. These operations are part of the Request Retry module. During this course, the client regains all the necessary information for future request processing. All the requests that had been kept pending during this course are then served and the CFS-MDS transaction continues in the normal manner.

The primary server may fail again; and as long as a backup server is provided, it would failover and a highly available metadata service provided to the client.

3 Design

This section elaborates on the design goals mentioned briefly in the previous section. It describes the architecture and functionality of the HA-MDS framework.

3.1 Failure Detection

This component deals with detecting the existence or 'liveness' of the Metadata Server. Two daemons are used in parallel to detect server failure:

- Heartbeat daemon [10] is used to detect whether the server is responding to network connections.
- Shared-storage daemon is used to detect whether the server is accessing shared storage.

Heartbeat Daemon

The heartbeat daemon examines server failure over the network. "I am alive" messages are exchanged between the primary and backup Metadata Servers. These messages are the 'time_of_day' value on each system. Three consecutive timeouts signal failure. The backup Metadata Server can detect failure of the primary Metadata Server in **9 seconds**.

Shared-storage Daemon

The shared storage daemon checks for server failure over shared storage. The daemon on each server writes the 'time_of_day' value and reads the value written by the other server after regular time intervals. If a difference of *8 seconds* is observed in the value read and the expected value, then a timeout has occurred, causing a check on the failure of the other server. Two consecutive timeouts signal failure. Thus a server can detect failure of the other server in **16 seconds**.

Service Manager

The functionality of the daemons differs on the primary Metadata Server and on the backup Metadata Server. Thus a manager is required to start the daemons for the appropriate configuration. For this purpose, the service manager is used, which reads a file on the shared storage to detect presence of a server. It reads the file five times; if the value changes even once then it is configured as MDS2, else MDS1.

3.2 Failover Component

The task of this component is to transfer the state from the primary Metadata Server to the backup Metadata Server. This includes the following functionality:

- Force failure
- Exchange IP Addresses
- Transfer state
- Restart Failure Detection component

The primary Metadata Server could fail due to network failure or system crash. In the case of a network failure it implies that the server is unable to reach the network but is configured as a Metadata Server. For such a scenario it is required that the backup Metadata Server force the primary Metadata Server to *sync* (flush) its data onto shared storage and kill itself. This concept is commonly called *Shoot The Other Node In The Head (STONITH)* [11]. The secondary Metadata Server comes up by forming into its memory all state information and structures that the primary server was maintaining. These structures and state information are obtained from the last received file in which the primary server maintains the in-memory state information.

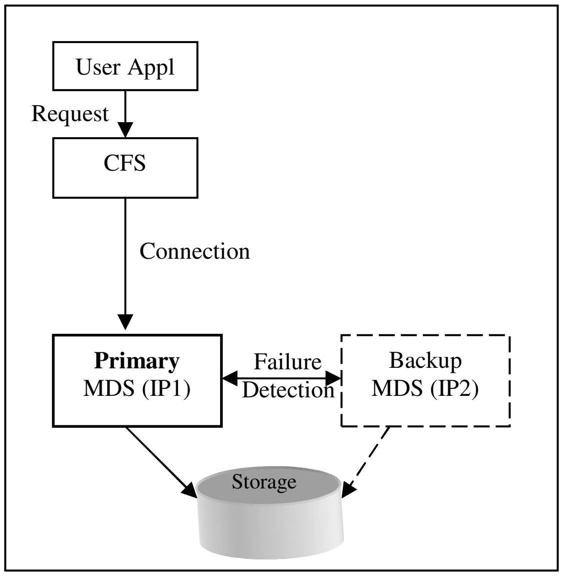

Fig. 3: Before Failure

The IP addresses of these two Metadata Servers are exchanged. The network interface card on each system has a unique Ethernet address. To affect an IP exchange, the Ethernet address mapped to MDS1's IP address needs to be changed to MDS2's Ethernet address. *Address Resolution Protocol (ARP)* [13] cache flush is used to change this mapping.

The backup Metadata Server then reads the shared storage for the client transaction information and system metadata. The Failure Detection component is then invoked for this new MDS1 configuring the backup Metadata Server as MDS1.

3.3 Request-Retry Component

Clients establish connections with the server using a socket[1]. Over this physical channel, the client can establish several logical connections. In the case of a timeout, an upcall is invoked to establish a new physical connection in user space. If the timeout had occurred due to failure of MDS1, then the failover mechanism will activate MDS2 in the time span during which the request will timeout. Hence a replaced Metadata Server is made available before the client tries to establish a connection with it. To establish this physical connection an upcall is made to the user space. Over this physical connection, a logical connection needs to be made to recover address of state information for the client.

Figure 4 and 5 show the system operation view before and after failure of the primary Metadata Server.

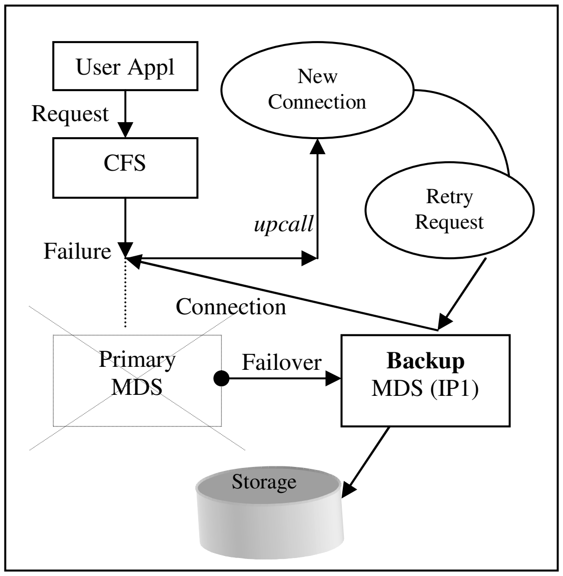

Fig. 4: After Failover

For each client that the Metadata Server connects to, it maintains in its memory, as well as in the last-rcvd file a structure that informs about the client's identification and the last transaction that it has successfully performed. Although the backup

[1] Socket: communication endpoint

server, during failover brings up these structures in its memory, the client needs to know about their address in memory. Considering that the frequency of metadata operations is much less than the frequency of data operations, we can conclude that only a few clients would urgently require a recovery from a primary server failure. Thus, this reconnection is handled by a recovery daemon on the client subsystem.

4 Characteristics of the System

The HAMDS system will prevent outages due to the following faults:

- Network card failure of the main MDS
- Ethernet failure of the main MDS
- Processor failure for main MDS
- RAM/Cache failure for main MDS
- Buses, battery, fan for main MDS
- File System / OS error
- Software design error
- Hardware design error
- Power failure
- Loss of cooling due to hot and unfavorable conditions
- Inexperienced/malicious user/operator causing main MDS to crash
- Software installation/upgradation
- Upgrading the Hardware (except SCSI)

With considerations to the above-mentioned cases, the downtime in a year will be *513.208 minutes*, i.e. ***8.5534 hours*** [25]. Thus, the solution will provide 99.9% availability. The HA-MDS for Lustre system belongs to **Class 3**.

5 Test Results

The Metadata Server in the existing Lustre system is a single point of failure. Any requests sent by the clients after the Metadata Server crashes will timeout and result in a system failure.

From the tests carried out, we conclude that the backup MDS takes over within 30 seconds after primary MDS failure. Request times out after 30 seconds after being sent. New connection is established. System operation continues un-interrupted. Thus, the client recovers. The HA-MDS provides un-interrupted service to Lustre clients for metadata transactions.

We have also tested the performance of Lustre and our HA-MDS system using IOZone; a file system benchmark tool. Figure 5 illustrates the performance comparison between lustre, chirayu_lustre (HA-MDS) and chirayu_timeout (after failover) [26].

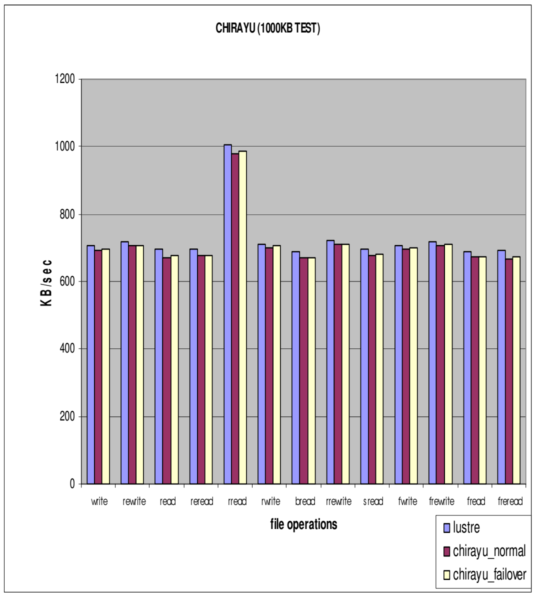

Fig. 5: IOZone Test Results

6 Conclusions and Future Work

As the need for scalable cluster systems increases, server failures become extremely intolerable. Lustre is an example of a cluster file system supporting 10000's of clients managed by a single Metadata Server. This makes the server a single point of failure. We have designed and implemented a Highly Available solution for the Metadata Server in Lustre, belonging to Class3.

The system, which we have built, is highly available. The clients get continuous service but they are interrupted for the failover time, which is approximately 30 seconds. We now aim to make a ***fault tolerant*** system in which there is no waiting period for the clients, i.e. instant failover. This will be achieved by forcing the backup server to build up state simultaneously with the primary server by listening to all client requests. It does not wait for the primary server to fail before it starts building up state information about the clients.

The other consideration is that the backup server in our system is idle, i.e. it only monitors the primary server and does not act as a server. An enhancement to this would be to enable the backup Metadata Server to service client requests for different namespace. This would remove the overhead of an idle server, as the backup server would also be functioning as a Metadata Server.

Appendix A

MTBF and MTTR of system components

Outage	MTBF (hours)	MTTR (minutes)
Realtek NIC card [17]	50000 – 100000	15
CAT5 Ethernet Cabling	100000 – 200000	10
Fast Ehernet Switch [18], [19]	200000 – 500000	60
Processor [24]	70000-100000	500
RAM/Cache [20]	200000 – 500000	60
SCSI disk [21], [22]	100000 – 300000	300
Linux Crash [23]	4360 – 8760	240
Data Organization on SCSI	4380	120

Acknowledgements

We would like to thank Dr. C.V.K. Rao, Head of Computer Department, PICT and Mr. Aditya Kini from CalSoft. Pvt. Ltd. for their guidance throughout the study of Lustre. We are also grateful to Mr. Phil Schwan, Cluster File Systems, Inc. for his words of encouragement, Mr. Anandamoy Roychowdhary from CalSoft Pvt. Ltd. who helped us throughout the year. A special thanks to Mr. Amey Inamdar and Mr. Kedar Sovani from CalSoft for their brotherly support during the development work on this project.

References

[1] Peter J. Braam, 'The Lustre Storage Architecture', Cluster File Systems, Inc.
[2] Thomas M. Ruwart, 'OSD: A Tutorial on Object Storage Devices', Advanced Concepts Ciprico, Inc.
[3] Gene Milligan, 'Information Technology - SCSI Object Based Storage Device Commands (OSD)', T10, Working draft, Seagate Technology Inc. NCITS TBD-200X, Project 1355D, Revision 3, 1 October 2000.
[4] Mike Mesnier, 'Rebirth of Object Based Storage', Intel Labs, January 22, 2002.
[5] Peter J. Braam, 'Lustre: A Scalable, High Performance File System', Cluster File Systems, Inc.
[6] Peter J. Braam, 'Lustre: A High-Performance, Scalable, Open Distributed File System for Clusters and Shared-Data Environments', Cluster File Systems Inc.
[7] Rumi Zahir, 'Roadmap Proposal for Lustre Request Processing & Bulk Data Movement', Intel Labs, Intel Corporation, June 3, 2002.
[8] Gregory Pfister, 'In Search Of Clusters', 2nd Edition, Prentice Hall Inc., 1998.
[9] A.K. Bhide, E.N. Elnozahy, S.P. Morgan., 'A Highly Available Network File Server', In Proceedings of the USENIX Winter Conference 1991, pp. 199-205, Jan 1991.
[10] Alan Robertson, 'Linux-HA Heartbeat System Design', SuSE Labs, Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta Atlanta, Georgia, USA October 10 –14, 2000.
[11] Alan Robertson, 'Resource fencing using STONITH', IBM Linux Technology center, Colorado.
[12] W. Richard Stevens, 'UNIX Network Programming', Prentice Hall India, 1999.
[13] W. Richard Stevens, 'TCP/IP Illustrated, Volume I The protocols', Pearson Education.
[14] Daniel P. Bovet, Marco Cesati, 'Understanding the Linux Kernel', O'Reilly Publications, March 2002.
[15] Maurice Bach, 'The Design of the UNIX Operating System', Prentice Hall (1987).
[16] Roger S.Pressman, 'Software Engineering- A Practitioner's Approach', Fifth Edition, McGRAW-HILL International Edition.
[17] http://adaptec.com, Adaptec Duo64, "ANA™-62022 Two-Port, 64-Bit PCI Network Interface Card for Fast Ethernet Environments."
[18] http://www.sixnetio.com/html_files/products_and_groups/mtbf.htm, SIXNET MTBF for fast ethernet switches.
[19] http://www.starmicrotech.com/index.cfm,Cisco Catalyst 2950G-24 24 port Intelligent Ethernet Switch.
[20] http://www.synchrotech.com/
[21] RK05/DIABLO/PERTEC DRIVES UPGRADED TO SCSI AEM-5C Cartridge Disk Replacement
[22] http://www.hardwareanalysis.com/
[23] http://gnet.dhs.org/stories/bloor.php3, Bloor Research, "Why Linux is better than Windows".
[24] http://www.itox.com/pages/products/mothers/370/gcs15.cfm
[25] Sapna Bafna, Shruti Dalvi, Abhinay Kampasi, Aditya Kulkarni, 'Increasing current Lustre availability to 99.9% with a backup Metadata Server', Jan 2003.
[26] Sapna Bafna, Shruti Dalvi, Abhinay Kampasi, Aditya Kulkarni, 'IOZone Test Results for HA-MDS', March 2003.

■ ■ ■