Amin Arab

Matt Roberts

Nathan Whitaker

Vineet Amonkar

# Metadata Server Project (Team C)

This project was to implement three different mechanisms for storing a metadata server. The metadata server is responsible for implementing a filesystem on top of an existing object store. This means that reading and writing to files is handled by the object storage engine but the part that our team implemented managed, creating directories, creating files in the metadata server, removing files, removing directories, and listing flies. (The corresponding commands of LS, MKDIR, TOUCH, RM, RMDIR in a unix based filesystem)

This project was implemented in three ways, with a Centralized server, with Distributed Servers similar to Ceph MDS, and using a general-purpose distributed database as the backend (Cassandra)

The additional components implemented were a lockserver, command line client, client simulator and a request generator. The lockserver was responsible for maintaining locks for the metadata stored in the general purpose database, (a similar component was implemented in Ceph for Cross MDS locking). The command line client makes it possible for people to enter commands for the servers to run. The client simulator reads in a file (containing a large or small directory structure tested with files up to 4gb) and then walks down the parsed directory structure, creating directories and files. Optionally the client simulator also has the ability to do randomly walk down the tree created from the file. The Request Generator makes random requests to the server and it is used for measuring the performance of different servers.

# Centralized and Ceph Server

Both the Centralized and Ceph servers are HTTP based and respond to common Directory manipulation commands through the /command Resource. The Metadata is stored in memory in a Tree Structure and operations submitted to the server manipulate that tree structure. Each Node in the Tree has a ReadWrite lock that ensures that concurrent requests don’t interfere with eachother, leaving the server in an invalid state. The only difference between the Centralized and Ceph Server is that the Ceph Server has an additional feature to enable the Tree to be “Partitioned” this partitioning takes a single directory and moves it to a different process (on the current server or on a different one). This is for scaling if the metadata information or the request load becomes larger than a single server can handle.

The Ceph Server has an additional component that responds to a custom TCP protocol, this is to ensure that if a server hosing a subpartition of the tree needs to take out read locks on directories hosted on a different server. This custom tcp protocol is based on the lockserver implemented for the general purpose database solution.

Here is the UML Diagram that outlines the important classes in the Ceph and Centralized solution.



Simplified Class Diagram

# Ceph Client

When the Server detects that the node on the path resides on another server, it returns a Redirect message with the address of that server. The Ceph client builds a structure to associate this node with the server containing the partition. Any future request that is made with a path that contains this node would then be routed to the correct server. The ceph client utilized longest prefix matching using Trie data structure to match the path with the correct server.

# Unix Solution

Unix Solution is the name given to the method of using a general purpose database to store all the directory information, as this is similar to the unix methodology of “everything is a file”. The information is stored as a text blob in the database with the filename as the key.

Since the component that stores the information is a distributed database, all of the logic for interacting with the metadata had to be programmed into the client, additionally there’s nothing stopping anyone with access to the client from maliciously modifying the data in the database.

Additionally the lockserver could potentially become a bottleneck as it has to respond to all requests, with a single thread being taken out per request.   


Performance

Initial Step:

We first built a tree by parsing a file. The file consisted of about 400,000 nodes(both file and Dir Nodes). The tree consisted of four very large directories about the same size of nodes (100,000) and couple of smaller directories.

Next we ran the request generator. The request generator plays the four operations(MKDIR,LS,RMDIR,TOUCH) with uniform distribution at a depth greater than four.

The depth requirement of the request generator is to ensure that the top level directories are not impacted my RMDIR which could drastically reduce the size of the tree and at the same time ensuring all operations are performed uniformly.

The tabulated results represents an average taken over all the clients.

Settings Summary

Size of Prebuilt Tree: 400,000 nodes

Number of Clients: 48

Number of Requests(all operations) by Client:1000

Depth Range : 5-8

Centralized Storage

|  |  |  |  |
| --- | --- | --- | --- |
|  | **MIN(ms)** | **AVERAGE(ms)** | **MAX(ms)** |
| **LS** | 0.413054 | 3.391712 | 1513.504650 |
| **MKDIR** | 1.111009 | 5.795678 | 82.896213 |
| **TOUCH** | 1.722967 | 5.319385 | 103.406056 |
| **RM** | 1.751115 | 5.573788 | 47.571984 |
| **RMDIR** | 0.931405 | 5.305296 | 106.921560 |

Ceph Storage

For the Ceph Storage, the four largest directories as stated before were distributed over the four servers before running the request generator

Additional Settings

Number of Servers: 4

|  |  |  |  |
| --- | --- | --- | --- |
|  | **MIN(ms)** | **AVERAGE(ms)** | **MAX(ms)** |
| **LS** | 0.641598 | 5.067741 | 533.714807 |
| **MKDIR** | 1.529790 | 8.147797 | 66.115379 |
| **TOUCH** | 1.692248 | 7.924746 | 176.404410 |
| **RM** | 2.249456 | 12.109834 | 147.450659 |
| **RMDIR** | 1.629728 | 7.424451 | 102.057709 |

Cassandra storage

We tried to go with the same tree length but it was taking a long time to build it using Cassandra operations. Therefore we had to change the size of the tree.

Settings Overridden

Size of Prebuilt Tree: 40,000 nodes

Number of Clients: 48

Number of Requests(all operations) by Client:1000

Depth Range : 3-5

Additional Settings

Number of Servers: 3

Replication: 1

Number of LockServers: 1

|  |  |  |  |
| --- | --- | --- | --- |
|  | **MIN(ms)** | **AVERAGE(ms)** | **MAX(ms)** |
| **LS** | 1.378253 | 6.755787 | 575.999250 |
| **MKDIR** | 1.027119 | 5.019692 | 35.051343 |
| **TOUCH** | 0.888927 | 6.344912 | 68.975634 |
| **RM** | 1.924324 | 8.417549 | 32.024492 |
| **RMDIR** | 1.980665 | 8.535870 | 144.664141 |

Performance Analysis

Out of the three types of servers, the centralized version performed the best following by Ceph storage and Cassandra Storage. The Cassandra Storage performed equally as Ceph but Cassandra Storage tree was ten times smaller than that of Ceph.

The Ceph server operations are generally 2-3ms behind of centralized server except the RM request which is unusually almost twice as expensive. We could not determine what is the cause of this.

However, as whole we believe the 2-3ms time difference is due to the network delay when there is locking across servers. We have determined that partitioning is not helpful for performance reasons but it is critical when there is a need to distribute the storage across servers.

The Cassandra Storage performed the worst by a huge margin due to it persisting data on the disk. It was also observed that longer the paths the expensive the operation. This could be attributed to the fact that each request to Cassandra returns one node and thus the longer the path the more requests are made to determine the directory structure to traverse.