CMPE 275 Project 1

Gossamer

(A distributed Storage System)

**Team Members:**

Ashutosh Singh

Viraj Nilakh

Shruti Padmanabhan

1. **Technologies used:**

|  |  |
| --- | --- |
| Languages | Java |
| Core Packages | Netty, Google Protobuffer |
| Databases | Redis |
| Builder | Apache Ant |

1. **Main Components:**

Client : Responsible for sending read and write requests

Server: Responsible for responding to messages from the client, participating in leader election

Leader: Responsible for replying to client requests

Follower: Store and send data

1. **Topology:**

Mesh Topology is being used as it was easier to build. Even in case of multiple node failures, alternate path could be found. Addition and deletion of nodes can be done without affecting other nodes.

1. **Workflow:**

**Two types of requests:**

1. **Write Requests from client connected to cluster**

Client sends a write request to cluster requesting leader to distribute the file across the network.

The first step involves client chunking the file and storing it in an in-memory Array List. This arrayList is then converted into Command Request messages which are then passed to an executor service which spawns as many threads as the OS allows and writes these messages to the channel to be sent to the leader.

1. **Read requests from client connected to cluster**

Client sends a read request to leader requesting a particular file by name. Leader replies back with the file chunks and client sorts these chunks and assembles them to a file.

1. **Read request from client requesting files saved to cluster**

Client send a requestAllFilesNames request to cluster to get a list of all files stored on the cluster.

1. **Leader Election:**

Cold Start:

When there is only one node in the cluster, then it becomes the leader. When another node joins this cluster, it starts receiving heartbeat from the leader. When the leader goes down in this case there is a reelection and new node becomes the leader.

All nodes start at once:

When all the nodes start at once, they first get connected to each other and then election takes place. When another node joins this cluster, it starts receiving heartbeat from the leader. When the leader goes down in this there is a reelection and new node becomes the leader.

Two nodes turn Candidates altogether:

In this case the node whose timer times out causes the second node to become follower and cast the vote for him. If one node is at lower term then the other then it becomes the follower and casts vote.

Vote request to Leader:

When a node asks for a vote from the leader, the leader provides his response of him being alive and turns the candidate back to follower state. This happens even when the new node joins the cluster for the first time with and active leader. But if the leader is at a lower term then the candidate then it becomes the follower.

Leader heartbeats + follower heartbeats

Both Leader and followers send heartbeats to each other. Getting heartbeat from leader, causes follower’s timer to reset and thus preventing reelections. Getting heartbeats from followers, prevents them from being deleted from the cluster and causes resetting of their timer

Dynamic Node addition

As soon as a node boots up, it is automatically added to the cluster and connected to all the nodes in that cluster. If a leader is present, it starts getting heartbeats from the leader. If the leader is not present an election takes place.

Dynamic Node deletion

When a node stops receiving heartbeats from any other node for a particular timeout period, that node gets deleted from the cluster dynamically.

Our approach:

To use 3 replicated redis servers for node registration, as soon as a server boots up. When a new node is registered, all the nodes become aware of that and try to connect to the new node in a mesh topology. When a node is deleted from redis, all other nodes become aware of that and they delete its entry from their connection list.

Our novel idea of using redis for node discovery was accepted by the class for getting the details of their next cluster Id in a ring topology

Difficulties faced:

* Whole of the semester time, seemed to be wasted, trying to debug raft algorithm when the problem was with the machine’s firewall causing the raft implementation to fail constantly. After turning off the firewall, all of the raft implementation worked swiftly as expected.
* To make redis server accessible by other hosts from different machines, its configuration file needed to be changed.
* Sometimes the host was unable to connect to redis and the server used to crash constantly without us figuring out the reason why. So we created a PingRedis class which will test redis connections for all the servers beforehand, eliminating the of pain restarting the servers and assuming that redis connection will be successful the next time

**Work Stealing**

We observed that, as we are already replicating files chunk by chunk across all the followers, each follower does equal job process write request. But only for the case of a read request it becomes an overhead only for the leader, as we are serving all the read requests to be performed by the leader. So inorder to relieve leader from most of the pain of read request, we use work stealing on read requests. Whenever a leader receives a work steal request, it will update its stealRequest variable to true, thus ignoring all the steal request further received until that stealRequest is served and reset to false. The followers are programmed to issue a steal request after every 7 seconds so that all the nodes would get equal chances to process the read request and only one cluster would not be overwhelmed by that.

1. **Replication**

We are using W=1 replication as it would be easier implementing work stealing as all nodes would have the data and any node can steal message request from a queue of any other node. We are using Redis nodes at each cluster node to keep track of all chunks.

1. **Node discovery**

**For a period of 30 seconds, after the cluster formation is initiated, all nodes broadcast to all other nodes. This allows initial ode discovery and output is written to a route.conf which is used to form edges between nodes.**

**After the initial timer of 30 seconds has expired, the connection thread shuts down and leader election takes place. From now on, only the leader can add nodes to network. Any new node wanting to join the cluster sends a connect request to leader and leader informs other nodes of the new node and connections are formed.**

1. **Client API**

Client allows four options to the user.

**Option 0 :** Exit

**Option 1:** Write file to server

**Option 2:** Read all file names from server

**Option 3:** Read a specific file from server

1. **Knowledge gained**
2. **Netty**
3. **Redis**
4. **ProtoBuff**
5. **Architecure advantages and disadvantages**
6. **Leader election algorithms**
7. **Java multithreading handling**
8. **Contributions:**
9. Ashutosh Singh: Node discovery, Client API, Write by multithreading, Write to leader by messages, Refactoring, Report creation, Testing, Research
10. Viraj Nilakh: Leader election, Replication to followers, Testing, Research
11. Shruti Padmanabhan: MySQL adapter, Research, Testing