Luis Remis - remis2
Xuewei Zhang - xzhan160

Design and Implementation

1. Membership Control and Leader Election

Our membership control protocol adopts the concept of SWIM protocol. If no machine fails, the message load is small (always O(1)). Since infection-style dissemination will increase the message load when no one fails, we use TTL broadcast to do dissemination. We use our membership protocol as a base for our MP3 implementation. We implemented the bully algorithm for the leader election, which proved to be very fast. Since we implemented it using UDP, we had to add extra checking in order to detect erroneous election. We choose to use UDP so that we could reuse much of the implementation of the membership protocol, and we wanted to encapsulate the election and membership in a single building block. We use Log-Querying program from MP1 to track UDP message in the system.

2. Virtual Ring Style Distributed File System Design

For the file system we implemented a virtual ring by hashing the IP address of the nodes. We chose this approach since having a master would mean a single point of failure, and the administrative cost of having replicas of the master's information would be equivalent as implementing a non-centralized protocol. And by using the hash of IP address as the location, we can make nodes more distributed on the ring.

This way, we have a **fully decentralized file system**.

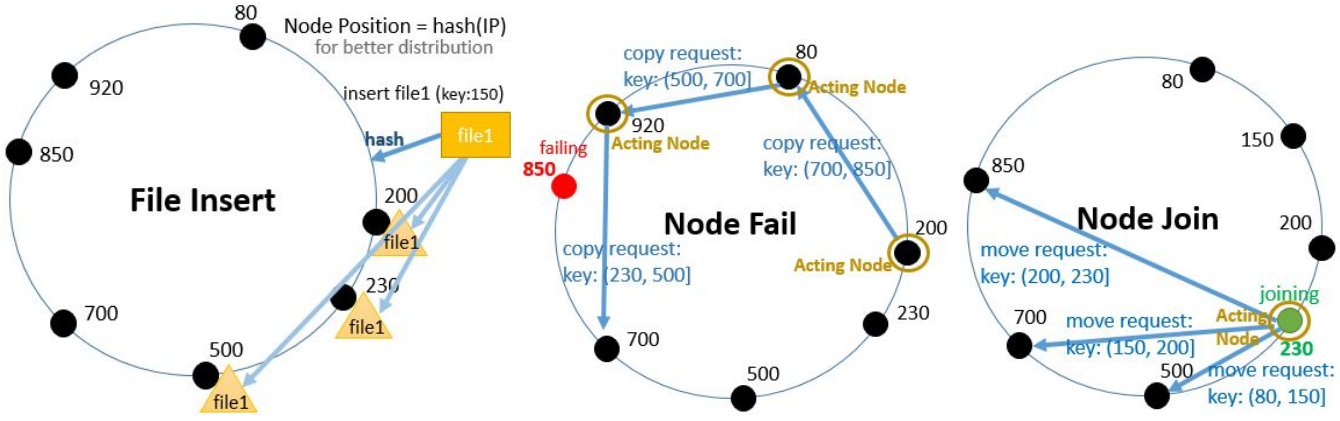
The files are located in three different machines according to the hash of its name (its name in the filesystem). Every new file name is hashed, and according to the result, is sent to a master replica and two secondary replicas. The hashing method is very convenient since it can be used in all the machines that belong to the system, and thus, knowledge of every file and its location can be found in every machine in O(1).

3. File Manipulation for Leaving or Joining

Using the membership protocol, the file system knows about the presence of all the other nodes, and has information about the virtual ring. To maintain encapsulation and abstraction, we implemented a message queue to talk between membership protocol and file system, and using conditional variable to prevent busy waiting.

Whenever a new node joins, every other node is notified to update the ring. And the new node ask for the files that belongs to its position in the virtual ring from the neighbor nodes. The new node is in charge of letting other nodes which files are needed.The new node does this by sending request to neighbours to move the files.

In a similar way, when a node fails, all the other nodes are notify and take action, knowing the new configuration of the ring and knowing which files should be requested from other nodes. Because of this, both the join and the leave (or fail) have similar behavior: Nodes are responsible for asking for files belonging to them and for notifying other nodes that these files must be moved from them. Thus we can keep every file has three replicas, even at two node simultaneously fail or join.



Luis Remis - remis2
Xuewei Zhang - xzhan160

4. Distributed Grep for Debug

Our MP1 implementation was very useful during all the testing and performance analysis since we could concentrate all the results in a single machine and have an easier analysis of what was going on in the system as a whole. We could check all the logs from different machines which contained all the information we needed to build our graphs and analysis.

Performance

We run a series of experiment in order to test the performance of our implementation.
The first experiment was testing how long it takes to elect a new leader when the following situation arose:

- When the actual leader leave voluntarily: less than 1 second.
- When the actual leader fails: less than 2 seconds.
- When a new node joins with the highest id: less than 1 second.

The analysis of the bandwidth and latencies is the same af for the bully algorithm, since we implemented the algorithm discussed in class. In the case when UDP packets get lost, the algorithm is run again.

In order to test the latency of adding one file, we measure the wall time of the method that send a 20MB file to all three machine that are supposed to hold the file. On average, it took 31 ± 2 ms to place the file on all the machines. This can be optimized by opening connections in parallel with the three nodes. We plan to optimize the send/recv of files as a future work, using multiple threads to handle communication. **It is important to notice that our file system works on RAM disks, this is why the latencies are relatively small.**

For the read/write test, we use 20MB and 200MB to measure the latencies. The below plot shows our results. The analysis is complicated since we have no information about the underlying network. Assuming a 100Mbits connection, the theoretical transmission time for a 200MB file is 16 seconds, but our wall time results shows hundred of milliseconds. This may be because the underlying network is gigabit ethernet, or because the virtual machines are in the same node, and then network communication is negligible. On the other hand, the results seems coherent when comparing a 20MB and 200MB (the first case is almost 10 times faster in both reads and writes). The writing takes longer since it incurs in communication with multiple machines, different from the reads where the communication is with a single machine (master replica). Because of limitation in the ram space, we loaded 1GB of the wikipedia corpus in 2.2 seconds. The latency is bigger than a single 2GB file since multiple opening and closing communications are involved when having small files.