Midterm Project-Chord, Part 3: File System Layer Implementation

In Part 3 of the midterm project, you are required to implement the file system layer for your Chord-based distributed file system on AWS cloud.

You need to implement three functionalities in your distributed file system: data migration, file chunks, and replication. After implementing these three functionalities, you can also implement additional functionalities to earn extra credits for the assignment.

You are free to use any method to implement the functionalities you want to implement. You can use any programming language to implement them and add any components you need to the system.

Chord Implementation

In Part 2, you were provided with a binary of the chord node. In addition to the `"get_info"`, `"create"`, `"join"`, `"find_successor"`, and `"kill"` RPCs mentioned in Part 1, the binary also exposes the following RPCs:

- `Node get_predecessor()`: returns the predecessor of the node.
- `Node get_successor(int i)`: returns the i-th successor of the node. `get_successor(0)` returns the immediate successor of the node; `get_successor(1)` returns the successor of `get_successor(0)`; `get_successor(2)` returns the successor of `get_successor(1)`. The parameter `i` can only be 0, 1, or 2.

You may leverage these RPCs for implementing your distributed file system.

Requirements

We will evaluate your implementation based on correctness and efficiency.

You need to submit all the code you use, including the code for all the components in the system and the scripts you use. In addition, you need to write a report explaining the components in your system, the code and scripts you submit, and your system design.

1. Correctness - Data Migration (33%)

After completing Part 2, your Chord-based distributed file system should be able to automatically scale out a new cloud node and add it to the system when the storage usage of the entire system exceeds a certain threshold. The Chord node on the newly added cloud node should be able to join the existing Chord system automatically. When your client program uploads or downloads a file, it should first calculate the hash of the file name and then use this hash to call the Chord system's `"find_successor"` RPC to obtain the information of the cloud node responsible for storing the file, and then upload/download the file to/from the file server on this cloud node.

However, when a new cloud node joins the entire distributed file system, a new Chord node will also be added to the Chord system, and the successor of some file hashes may become a different Chord node. Therefore, to ensure that the client program can still successfully find the cloud node responsible for storing the file, your system needs to automatically perform data migration when a new cloud node is added, sending some files to the newly added cloud node.

Please note that if your Chord-based distributed file system fails to meet correctness, you will not receive credit for the subsequent parts.

2. Load Balancing - File Chunks (33%)

When a user uploads a large file to a particular cloud node, it may overload that node compared to others. To avoid this situation, we can divide the file into fixed-size chunks and upload these chunks to the cloud nodes responsible for storing them.

To achieve load balancing, you need to modify your client program to split the file into multiple chunks before uploading it. Similarly, when downloading a file, the client program should download all the file's chunks and then assemble them into the original file. The size of each chunk should be between 4 KB and 4 MB. If you choose a chunk size that is not in this range, please explain your reasoning in the report you submit.

3. Fault Tolerance - Replication (33%)

When a cloud node fails, the files stored on it may become unavailable. If the storage on the node is damaged, the files may become permanently unrecoverable. To prevent this, we can store the same file in N different nodes, ensuring that users can access all files in the system even if (N - 1) cloud nodes fail. Additionally, when a cloud node fails and the number of replicas of a file decreases, your system must be able to automatically replicate the file to restore the number of replicas.

Your system must maintain at least 3 replicas for each file. If your system only maintains 2 replicas, you will only receive partial credits. You also need to design experiments (for example, using `"kill"` RPC to stop some Chord nodes in the system) and demonstrate that your system can automatically maintain replicas.

If you are unsure how to proceed, you can refer to the system design of CFS to implement the replication functionality of your system.

4. Report (0%)

You need to submit a report (in any format) that explains your system design. Your report must include the following:

1. System components: Which components run on each cloud node, what are their functions, and how are they implemented (using existing tools or code you wrote).
2. System functionalities: How you use the components in the system to achieve Data Migration, File Chunks, and Replication functionality, and why you chose this design approach.
3. Experiment: You must design simple experiments to demonstrate the functionality you have implemented and measure your system's efficiency.
4. All references you have consulted.

You **must** submit a report, and your work will be evaluated based on the report.

5. Bonus (33%)

After completing all the requirements, you can add additional features to your system to obtain up to 33% (equivalent to a 3% semester grade) of extra credits.

To receive extra credits, you must first complete the assigned system features: Data Migration, File Chunks, and Replication. In addition, the new feature you add must involve improving the system architecture. It is recommended that you discuss your design with TA before implementation. Like the other features, you must explain how you use the components in the system to achieve the new feature and design experiments to demonstrate it in your report.

Here are some suggested features:

1. Implement caching to reduce the burden on cloud nodes that hold popular data.
2. Implement Deletion functionality and **scale in** the system based on storage usage.
3. Implement virtual nodes to distribute files based on the different capacities of cloud nodes.

Submission

Please put all your files together with your report in a folder named `<Student ID>_chord`, compress the folder as `<Student ID>_chord.zip`, and submit it to COOL.

```
<Student ID>_chord/
|
+-- report.pdf
|
+-- src/
|   |
|   +-- (implementation.py)
|   |
|   +-- ...
|
+-- scripts/
|   |
|   +-- (script.sh)
|   |
|   +-- ...
|
+-- ...
```

Late submissions will not be accepted.