SE347 - 2021 Spring

Lab 2: Naive gDocs

Due: 2021/7/15 23:59

Introduction

Lab-2 is an open assignment, and you can choose any programming language and technology stack you want. In this lab, you need to build a complete distributed application from scratch, which is an online document editing tool for multi-terminals cooperative (which is similar to Google Docs and Tencent Docs).

In this document, we call it ***"Naive gDocs"*** (Naive Google Docs).

Generally speaking, lab-2 can be divided into two parts:

1. You need to build a ***Distributed File System (DFS)*** primarily.
2. And with the support of DFS, you need to build the online document editing tool ***Naive gDocs***.

Please note that your Naive gDocs **MUST** be built on the basic of DFS. That is to say: you are **NOT** allowed to use other similar system to replace DFS.

Collaboration Policy

You must write all the code you hand in for this course, except for code that we give you as part of the assignment. You are not allowed to look at anyone else's solution, and you are not allowed to look at solutions from previous years. You may discuss the assignments with other students, but you may not look at or copy each others' code. The reason for this rule is that we believe you will learn the most by designing and implementing your lab solution code yourself.

Getting Start

It is similar to the steps in lab-1.

1. Find your teammates in the class and pick a leader. **The max number of students in a team is 4.**
2. Click the link: https://classroom.github.com/a/KhaOp8Tl.
3. If you are the leader of your team, create a new team (you can choose the team name you like). Otherwise, you are a member of team "A", and you should join an existing team named "A".
4. Then GitHub Classroom will create your teams' **private repository** in the SJTU-SE347 organization (whose name is "gdocs-your-team-name").
5. Clone the repo and start your lab.

> **Note:**
>
> When you finish this assignment, you should commit all materials to the repository. It should include:
>
> - The source code of your project.
> - Two reports: design report and testing report.
>
> In summary, after the submission, the structure of the repository of your team should be:

```
your-repo/
+-- README.md
+-- .gitignore
+-- code/
    +-- {your-project-code}/
    +-- README.md (Give a brief introduction to your project)
+-- docs/
    +-- Team.txt (All the members' (ID, name) and the workload of each member)
    +-- {Design Report}  (Introduce how you design the system)
    +-- {Testing Report} (Introduce how you test the system)
    +-- Lab2-gDocs.pdf   (The lab description)
```

Naive gDocs

Naive gDocs is an application similar to Google Docs and Tencent Docs. In this lab, you can imitate these two products to improve your gDocs application.

gDocs can be presented in any form, either APP or Web is OK. And here are the basic features.

TASK

You **ONLY** need to choose **ONE** document type from:

- "docs", which is similar to Office-Word,
- "sheets", which is similar to Office-Excel.

gDocs needs to implement the following basic features with the support of DFS:

- The file is in **plain text**. (Even though the file is in plain text, different text styles can be shown to users via front-end technology, such as "Markdown".)
- For the same file, **multi-users collaboration editing** is allowed. When one user modified the file, other users should see his/her modifications.
- When a user is modifying a certain location of the file, other users are not allowed to modify this location.
- Implement a **file recycle bin**.

Here are some advanced features, achieve one or more these advanced features and you will get bonus.

CHALLENGE

- Choose the document type **"slides"** (which is similar to Office-PowerPoint), you can refer to How to create a basic slideshow presentation in LaTeX (overleaf).
- Allow inserting **pictures** or **videos** into a document. And this presents some challenges to your distributed file system.
- Record the **modification log** of the document. The user who modified the file and the places he/she modified should be recorded. You can display the log in chronological order.
- To support **version rollback** . According to the timestamp of modifications, your application should save different historical versions of the document. Users can rollback to somewhere according to the modification time (which is similar to git).
- Other features you think are interesting and challenging.

Distributed File System

In this section, you need to implement a Zookeeper cluster and build a **distributed file system** based on it. You may refer to Google File System to get some hints on how to design your system. We suggest that you should read this paper before starting this part.

1. System Architecture

Generally, you should organize your system in **client/server** architecture.

The servers include two types of nodes, **Master nodes and Data nodes**. Master nodes use Zookeeper to manages metadata of the filesystem, and Data nodes are used to save file content. Each node should be deployed as a single process and communicates with each other use **RPC protocol**. Both types should have multiple nodes for fault tolerance.

The clients are outer process, provide file read/write interfaces for upper applications, and request Master nodes to get/update metadata and request Data nodes to get/update file content.

2. Basic Requirements

TASK

- **Operation**

  Your system should provide basic file system interfaces for applications, e.g., `create`, `delete`, `open`, `read`, `write`.

- **Chunk**

  Like GFS, you should split files saved in your system into multiple fix-sized chunks, and these chunks are scattered across Data nodes. Master nodes are responsible for maintaining file-chunk mapping. In the writing operation, your client splits the file into chunks, then asks Master nodes for Data nodes to save chunks and send chunks to corresponds Data nodes. In the reading operation, the client uses the filename to query chunks' location from Master nodes and then requests data nodes to fetch file content.

- **Replication**

  Each file chunk should have replicas in different Data nodes to provide fault tolerance. Each replica has a version number linked to it to tell whether it's latest or not. Data nodes should record each chunk's version. At the same time, Master nodes must maintain each chunk's latest version number to avoid using stale chunks.

- **Fault Tolerance**

  There should be multiple Master nodes and multiple Data nodes in your system. Depending on your consistency strategy, your system should tolerate a certain number of node down events.

- **Consistency**

  Since there are multiple replicas for each chunk, the writing operation involves updating chunks in Data nodes and updating metadata in Master nodes. You should use some strategy (e.g., 2PC) to provide consistency assurance.

- **Concurrency Control**

  Your system should support multi-client. You can use Zookeeper to do concurrency control.

- **Failure Recovery**

  Your system should log the metadata changes so that when a node is recovered from a crash, it can recover metadata by redoing logs. Note that metadata not only refers to filesystem's metadata maintained by Master nodes; for Data nodes, they need to record chunk's version number, which is metadata for Data nodes.

3. Advanced Requirements

CHALLENGE

- **Scalability**

  You can design your system to enable adding Data nodes dynamically.

- **Load Balance**

  You can spend time to make sure chunks are evenly distributed in Data nodes as far as possible. If your system supports dynamically adding Data nodes, you may need to transfer existed chunks into newly added Data nodes to achieve better balance.

- **Checkpoint**

  You can use checkpoints to avoid unlimited growth of logs. To achieve this, you should build checkpoint from metadata and persist it to disk periodically, and in failure recovery, you should use both checkpoint file and log to restore metadata.

- **Efficiency**

  After realizing the basic requirements, we hope your system has higher read-write performance.

- Any other challenging advanced features you can think out.

Tips

The lab is a team-work. A team should consist of 2-4 people. And it is okay to finish it independently if you want.

Here are some details you should pay attention to:

- Please DO NOT paste a lot of code into the report(s).
- In the design report, you should focus on your system design and your solutions to the features.
- In the testing report, you should focus on the testing scheme(s). You can consider it from the following perspectives: performance, correctness, defects and corresponding improvement methods.

> **Note:** Here are some tips.
>
> - If you choose the document type "docs", consider using the Markdown editor as part of your front-end.
> - In order to prevent you from spending a lot of time on developing the front-end, you are **ALLOWED** to use some existing front-end components, such as Showdown, Luckysheet and so on.
> - The communication between gDocs and DFS can be achieved via HTTP protocol.
> - **DO NOT** pursue perfection too much. Your primary task should be to complete the basic features as much as possible.
> - Please keep in mind: The programming assignments are larger and more open-ended than in other courses. Doing a good job on the project requires more than just producing code that runs: it should have a good overall organization, be well implemented and documented, and be thoroughly tested.
> - Enjoy your coding!

Scoring Policy

Please do not forget that submit a description of each member's workload in `your-repo/docs/Team.txt`.

- Reports 10%
- Project Defense 20%
- Naive gDocs 30%
- Distributed File System 40%