CIS 505 Final Project Report

Wen Zhong WENZHONG@SEAS.UPENN.EDU
Yilong Ju YILONGJU@SEAS.UPENN.EDU
Haoran Shao HAORSHAO@SEAS.UPENN.EDU
Gongyao Chen GONGYAOC@SEAS.UPENN.EDU
Qianyi Guo QIANYIG@SEAS.UPENN.EDU

1. Design Description and Architecture Diagram

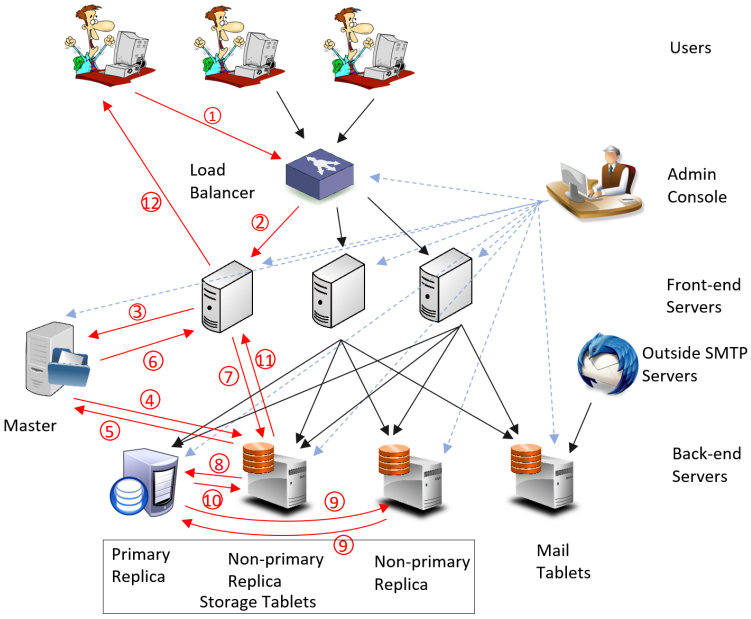

Figure 1. Overview of the system

The system is constructed with two ends, front-end servers and back-end servers that deal with human-computer interaction and storage process, and also a load balancer and a master node. When a user accesses the front-end server from the web page, the master node will direct the front-end server to the back-end storage server where the specific tablet of key-value is stored. Caching is used in the front-end server so that it will contact master node just once to get directed to the back-end server if it works well. Each time the user does a write operation, the back-end servers with the specific key-value stored should make replication in its tablet to avoid data loss.

A graph of the system architecture with workflow is shown below. The red arrows show the procedure of a write request to a storage back-end: 1, send write request 2, assign a front-end server 3, request the back-end server 4, check a candidate back-end server 5, confirm check 6, assign the back-end server 7, request write 8, send request to primary 9, replicate and confirm, if one back-end server crashes, write to log file 10, confirm replication success 11, response to front-end server 12, response to user.

2. Overview of the Features

2.1. Key-value store

The back-end system stores all the data is a distributed key-value store. This component has the following six features:

1. **Distributed**: We divided the tablets into more "task focused" tablets. There are three tablets for storage service, two for mail service and one for account service. This will make the load for servers to be more balanced. As services are also "distributed", for example, even if all mail nodes crashed, the storage service can still be used.
2. **Replicate**: Each value is stored on more than one back-end storage node to realize distributed partition. Primary-based replication is used and implemented by remote writes

method.

3. **Consistency**: As Primary-based replication is used as the protocol with remote writes method, each time a user sends a write request, he will get a confirmation once all the servers have been updated, or a failed response if there is something wrong. This ensures sequence consistency.
4. **Fault tolerance**: If a server launches a replication and its primary replica crashed and can't make a replication, the server can send a request to the master node and the master node will assign a new primary replica to the tablet and continue.
5. **Recovery**: All the servers has its own checkpoint log which frequently stores all its key-value pairs and other information on the disk. Also if the primary replica knows that a server crashed, it will write the PUT requests to a log file. When the server reboots, it first reads its own checkpoint file, then read the log file and download the corresponding data from the primary replica.
6. **Heartbeat**: All the back-end servers will frequently send a heartbeat message to the master node to confirm that it is working.

2.2. Master node

The master node keeps the mapping from ranges to tablets. The small tablets in each "task" tablets are divided by alphanumeric of the key. The master node has the following three features:

1. Gives clients the **mapping** to the back-end server when getting request and the client would contact the assigned nodes directly. The mapping is just the address of the back-end server with no actual key-value pair data. The back-end server is **checked** with two status. Both connection timeout and response timeout is checked to ensure both server and network reliance. Also, heartbeat from back-end server is accepted and that back-end server can be directly used.
2. Checks and assigns the **new primary** replica to a tablet if the old primary replica crashed. The back-end server is also checked for two status. Respond with the current primary replica to a rebooted server.
3. Responses the current primary to a server if it **reboots**. The master node is considered as "always working". It will reset the primary replica and clean all the logs when started. All the back-end servers should start after the master.

2.3. Load balancer

There are three load balancers in our system for user service, storage service, and mail service, respectively. Load balancers use round-robin scheduling to distribute workload on different front-end user/storage/mail servers.

When users visit the website for the first time, they will be asked to register/login through the user service. After they log in, the user service sets cookies for them, and then they can choose to enter mail or storage. If it's the first request, the user will be redirected to one front-end server using HTTP 301. Cookies just set by the user service will be transferred by the load balancer to front-end servers as well. Then the user can continue to use that server for future requests.

2.4. User account service

1. Support multiple user accounts. Different users could log in to the system because of cookie.
2. After a user logged in, show two buttons mail and drive, by clicking which user could get his own inbox and file folders.
3. Allow users to sign up for a new account, as well as change their passwords.
4. All user data are stored in the key-value store.

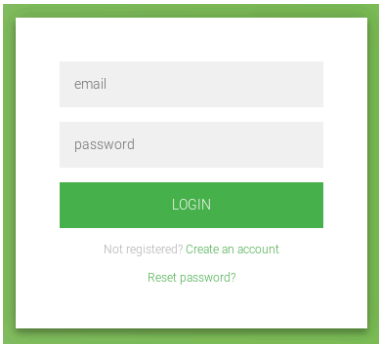

Figure 2. User login page

2.5. Webmail service

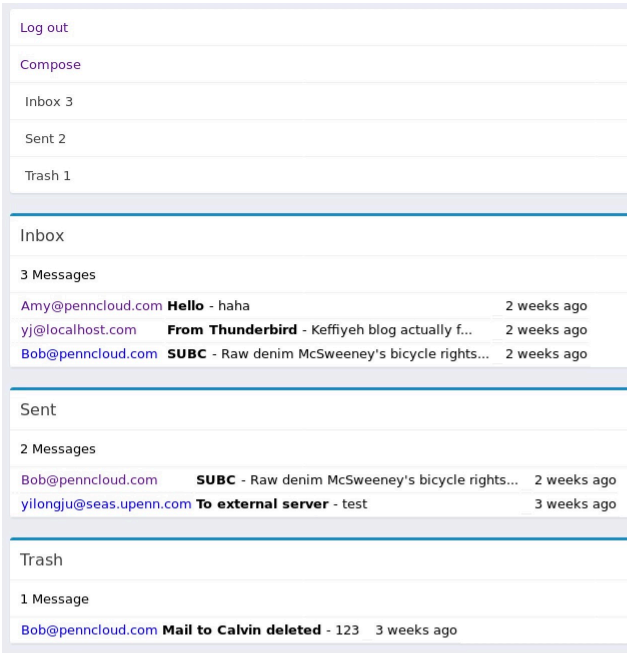

Figure 3. Webmail service with some mails

1. A page for the user to know numbers of emails in its inbox/sent/trash; a page to browse the emails in details, with Reply, Forward and Delete function; a page to compose new emails
2. All email data are stored on key-value store
3. Able to send/receive emails to/from `penncloud.com` users
4. Able to receive email sent from ThunderBird in local VM
5. Able to send email to external servers, like `seas.upenn.edu`

2.6. Storage service

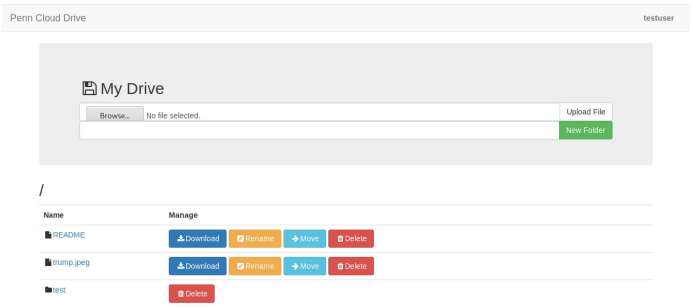

Figure 4. Drive with some files and folders

The storage service stores files as key-value pairs. It also stores serialized folder structure (the basic file system) in the key-value storage.

- In the key-value storage, the row key captures the information of file owner and file/folder path (format: `username@full_path`).
- For each row key representing a **file**, there's a *content* column that includes the base64-encoded binary content of this file. Currently, we have limited each value to be at most 1 megabyte to reduce network latency.
- For each row key representing a **folder**, there's a *items* column whose value is a serialized list of all folders and all files that belong to the folder (format: `[F@file_name, D@directory_name]`).

Finally, the service provides a web interface for users to manage files/folders in their drive. Users can upload, download, rename, move, and delete files. Users can also create and delete folders in their drive. Both text files and binary files are supported in this service.

2.7. Admin console

The admin page shows all the nodes in the system and their status. If a node is working, its status

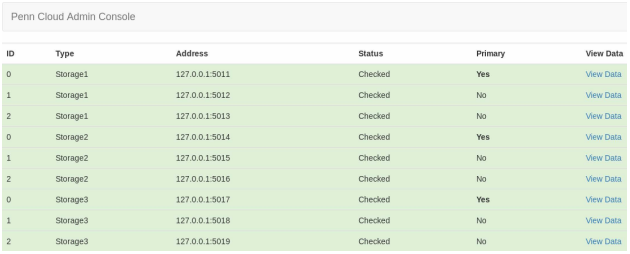

Figure 5. View node status in admin console

is "checked" and its background color is green. If it's not working, its status will be "connection timeout" or something else, and its background color is red. Primary nodes are also indicated using a bold "yes".

The admin console also allows the admin to view the raw data in key-value storage nodes by clicking "View Data" links above. Obviously, the link is available only for working nodes. A sample list of raw key-value pairs is shown below. If the value is too long to fit in the table (usually a base64-encoded binary file), it is shown as `[xxx bytes of data]`.

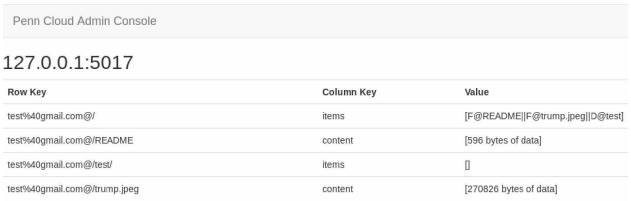

Figure 6. View raw data in admin console

2.8. One-key Start

We also implemented a convenient component (`penncloud`) that can launch and shut down the full system with one simple command in terminal.

3. Decisions and Challenges

1. Leader election vs. Master assign: We decided to use the master to assign the primary as leader election needs more messages and adds more uncertainty.
2. Client-centric vs. Sequential: As we used primary-based replication, it is easier to realize sequential consistency and aiming for orders for users.
3. Two-Phase vs. Add log: We decided not to use two-phase tolerance as we aim to let users always having a friendly feeling with as little failure as possible. So we want to make write and replication even if only the primary replica is working.

4. Responsibilities of Components

- User account service / Load balancer: Wen Zhong
- Webmail service / One-key start: Yilong Ju
- Key-value store / Master node: Haoran Shao, Gongyao Chen
- Storage service / Admin console: Qianyi Guo

References

Andrew S. Tanenbaum and Maarten van Steen. 2017. *Distributed Systems: Principles and Paradigms (3rd Edition)*. CreateSpace Independent Publishing Platform.