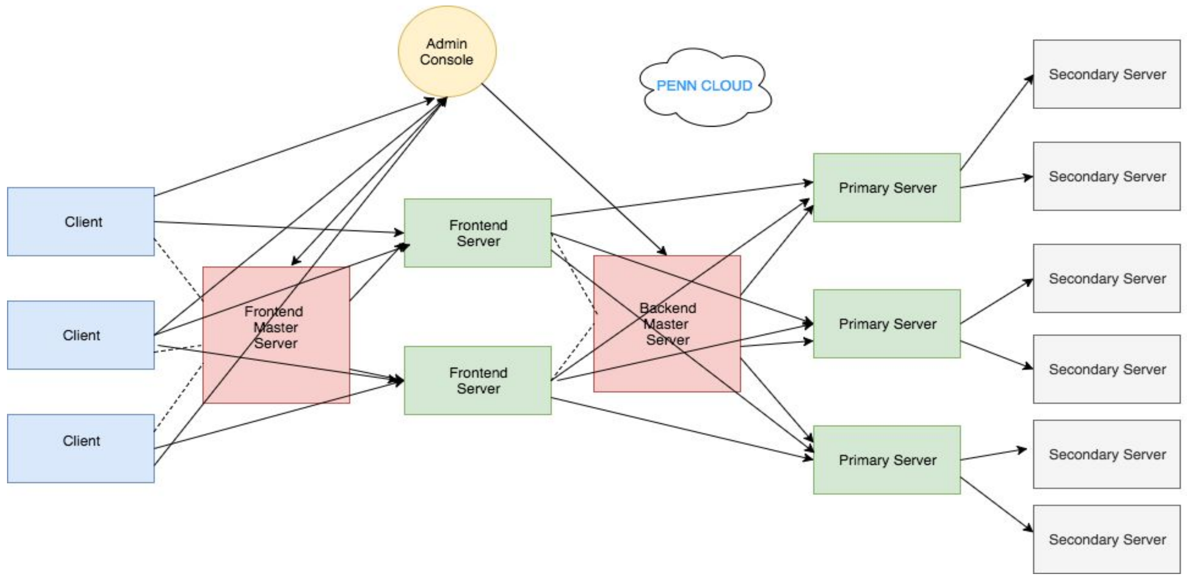
CIS 505 PennCloud Report

Team 5: Shun Li, Xi Wen, Yang Yi, Kexin Zhu

Overall Design



The frontend has one frontend master server, and two frontend servers. The backend consists of a backend master server, three primary servers with each of them connecting to two secondary servers. When the client opens the browser and enters the IP address, the load balancer will assign a port to the client. Frontend server client first connects with backend master server for load balancing.

When needing to query the backend key value stores, the frontend server contact backend master. Backend master server will send frontend server the primary tablet node port after dynamic load balancing. At the same time, primary tablet node will pass the same operation to the two secondary nodes for future backup. In addition, the client can also connect to the admin console which displays the status of all the frontend and backend servers and provides a way to disable the backend primary/secondary nodes.

The backend server uses TCP for all communications. For each tablet node, it will write each operation on the log file. Every 40 seconds, it will keep logging the operations like put and get and will snapshot the current key-value storage into bigfile, at the same time clear the log file. When a primary node crush, the client will timing out and reconnected with master, so that master will require the version of two backup tablet nodes and choose the highest version tablet node as the new primary tablet node. Then re-assign the client to the new primary tablet node. At the same time, the new primary tablet node will send the newest version backup table to the crush node for recovery.

For data storage, we use username as keys and columns are designed as follows:

- UID - username
- PWD - password

- M_HEADER_LST - list of email header data
- F_ROOT_LST - list of file header data

Each email and file has an uniquely generated ID (using library ulid), which will also be used as column name to store actual data of that node.

We also have one unique row key with name SID for session id, columns for this row will be session ids in browser cookie (also generated using ulid), and values in each cell will the user logged into that session.

Instructions of running code can be found here.

Features Implemented

Frontend:

- Register new user
- Log in
- Change password
- Send/View mail
- Upload / download / delete file
- Admin console
    - Admin console can display the status of the frontend servers and backend servers. It also provides a way to disable backend storage nodes by clicking a button, to test fault-tolerance. Admin console connects with backend master and frontend master to get a list of the status of all the server to display. When the disable button is clicked, the it connects again and refreshes the whole page.
- Load balancing server
    - The load balancing server assigns one of the frontend ports for to the client

Backend:

- Dynamic load balance
    - Masternode will record the clientname and the assigned server in a map, while servername and its group is recorded in a map at very first . So if one group is with less load, the new client will be assigned to the primary server of this group.
- Fault Tolerance
    - When a primary backend server down, the client will timeout and connect to backend master node. Masternode will detect the clientname(ip+port) and if it has been recorded in map, it will know the previous primary one was down. Then, send request to get the timestamp of two secondary node thus know the new primary. After the original is set on again, it will send to the new primary a command thus the new primary will copy its bigfile and send to the two replica now
- Communication between replica set
    - We manually set the primary tablet at the beginning, and the primary tablet will connect with the two secondary tablets
- Communication between master and whole tablet servers
    - Once the tablet node start, it will connect with master server

- Key value storage in binary format
    - Row and column as a key struct, value as binary format store in the map in the memory.
    - When backup, write the format (row,col): value into bigfile.

Major design decisions and major challenges

Design decisions

-Frontend
Master server listens on 8000
Frontend server listens on 8001, 8002
Admin console listens on 8020

-Backend
Master server listens on 8013 (for server), 8014 (for client)
Tablet server listens on 5001(Primary), 5002, 5003; 6001(Primary), 6002, 6003; 7001(Primary), 7002, 7003

Overall challenges

Since we have 1 frontend load balancer, 2 frontend servers, 1 backend load balancer, 9 backend servers, we have to open many terminals to listen on different ports, resulting the raising difficulty of connecting between frontend server and backend server and testing. Also, each of the teammates are using various versions of GCC compilers and environments, making the testing a bit harder and longer.

Challenges in frontend

For the SMTP mail relay, res_query() in resolv.h library cannot be compiled due to different versions of GCC. And there's a limited documentation about examples of using resolv.h for address lookup, so connections to the remote IP address using SMTP default port 25 is always failed binding.

Challenges in backend

For communication between one replica set, using TCP to communicate will be a major challenges for our fault tolerance. It will be difficult for us to handle the communication between the whole tablet set when one primary tablet node crushed. We need to consider which server be triggered first so it could be connected later by other node. Finally we designed a structure that primary will automatically connect with another two backup replicas, all the tablet nodes will connect with the master node when they are triggered. However, there will be problems when we implement the fault tolerance function, since our replica sets are treated as a whole, so we require the crush server to be restarted very fast. We have tried several methods, at first, we are using UDP to connect the replica set, and we accomplish the fault tolerance when the primary node crush, it can efficiently get to the new selected primary server data. However, we want to improve our performance since using UDP for communication will lose messages, so we change to TCP for the consistency, there is trade-off between consistency and availability in our design. In the end we have to give up our availability based on our assumption that the crush server should restart fast.

For the file storage, we first saved all the data in type of string. However, considering the images, we have to change all the data io into a binary way. The management of char pointer always went wrong and spent lots of time to locate the bug and fix it.

Work Distribution

Frontend Server (HTTP / frontend master) - Xi Wen
Frontend Server (SMTP / admin console) - Yang Yi
Backend Server - Shun Li
Backend Server - Kexin Zhu

Acknowledgement

Thanks for the efforts and contribution of all the teammates. Thanks for Professor Linh Phan and course staffs to provide us useful feedbacks and suggestion.

Appendix

Running Instructions

Backend first:
Run backend masternode: ./myprogram config2.txt
Run all the backend replicas(no primary): ./myprogram -s Secondary -p 5002 tablet_config2.txt
(5002,5003,6002,6003,7002,7003 available for replicas and the config is correspondingly tablet_config2.txt,tablet_config3.txt,tablet_config5.txt,tablet_config6.txt,tablet_config8.txt,tablet_config9.txt)
Run all the backend primary server: ./myprogram -s Primary -p 5001 tablet_config1.txt
(5001,6001,7001 available for primary and with config tablet_config1.txt, tablet_config4.txt, tablet_config7.txt)

Then frontend:
Code can be compiled by running make in frontend directory
Start frontend master:
./frontendmaster -p 8000 frontend_config.txt

Start two frontend server instances:
./frontendserver -p 8001
./frontendserver -p 8002

Start admin console:
./admin_console -p 8020