# Penn Cloud Project Report - T09

**Team Members -**
Jay Gala
Saurabh Raut
Shivani Rapole
Vandana Miglani

## Design Description -

### Backend -

Backend consists of 1 Master Server and several backend servers, arranged as 'server groups'. Each server group consists of servers, all of which have the same content replicated in their Key-Value stores.

#### Master Server -

The Master Server contains the following information -

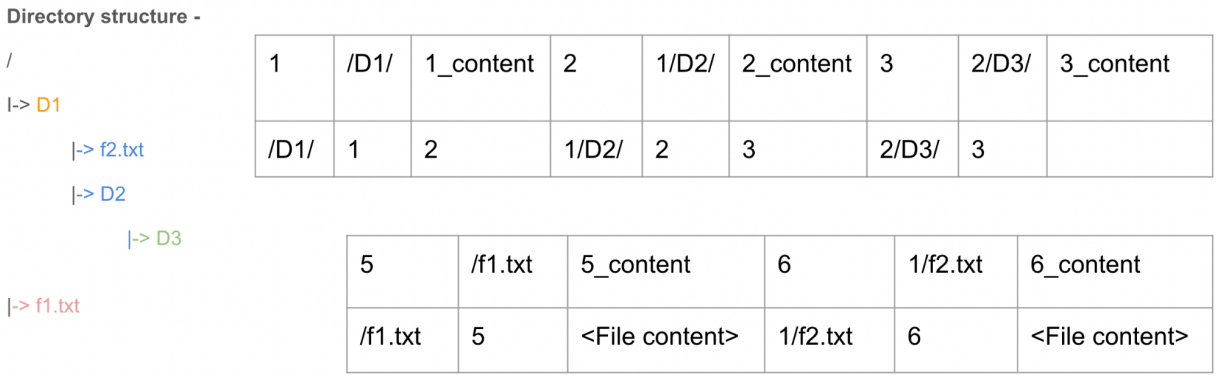
1. Mapping of the user to a tablet ID, which is in turn mapped to a group of servers that contain the particular tablet. Master Server chooses one among these servers to serve the request, based on the load on each server. Master Server will return the address of the backend server with minimum load to the front-end server.
2. Information about the active/inactive status of the server. The Master server also has a heartbeat checker thread which listens to heartbeats from all servers to determine their status. If the Master server does not hear back from a particular backend server, it is marked as inactive.
3. To recover from a failure, the master server uses the heartbeat checker. On detecting that a backend server has crashed, the Master server communicates this information to all other servers in that server group. Additionally, if the Primary server from a group crashes, Master allocates a new primary for the group and communicates this updated information to all the servers in the group. This helps the system handle failures elegantly.

#### Several Backend Servers -

The backend Servers contain the following information -

1. K-V store - The Key Value store supports 4 primary operations - GET, PUT, CPUT and DELETE.

Diagram of the key-value store -




Here, we have allocated counter values to each entry and maintained a bidirectional map structure. _content entries contain the associations with children nodes if it is a directory. If the entry is a file, _content contains the actual content of the file.

2. Each backend server also contains the address of all other servers in its server group as well as the address of the (current) Primary Server of the group. This helps in maintaining replication across all servers in a server group. On receiving a PUT, CPUT or DELETE command, a backend server forwards this request to the Primary server of its group. After receiving the request, the primary server then propagates this request to all the other servers in its group, each of which individually update their key-value store accordingly.

**Front end -**

Front-end consists of a load balancer and front-end servers each of which can handle multiple client requests.

Load Balancer -

Maintains the server status (active/inactive) of each front-end server along with the number of active connections present on each server.

1. It keeps on iterating through each of the frontend server to check if it is active or not and listens to the number of clients it is currently serving
2. Whenever a client (browser) makes some request, the load balancer redirects it to the frontend server that is active and has minimum load.

Front-end Server -

The front-end server handles requests from clients by making requests to the master/Backend server.

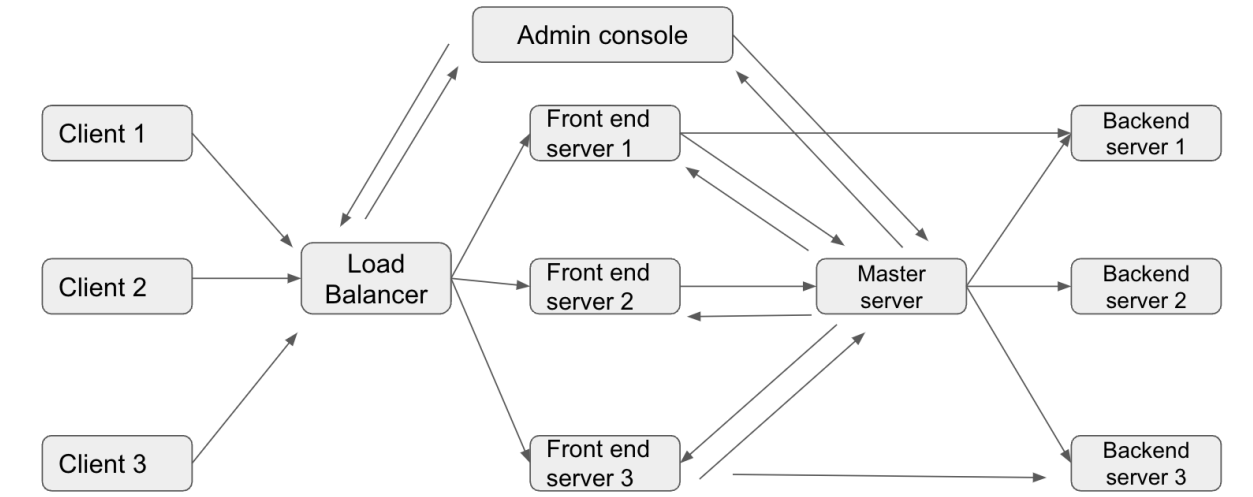
1. It is a multithreaded system which spawns a new thread for every new TCP connection it receives.
2. It then parses the HTTP requests of the browser, and determines which backend server to direct its request to by contacting the master server.
3. After receiving the respective backend server address, the frontend server would send appropriate commands to the backend server to satisfy the requests of the client. These commands are GET, PUT, CPUT, DELETE.
4. Hence, it also sends the client a proper HTTP response with the HTML page along with the requested content pertaining to the request sent by the client.

**Admin console -**

Maintains activity status of each front-end and back-end servers and can give commands to terminate any active backend-server

1. Gets the activity status of each front-end and backend server from the load-balancer and master respectively
2. When an user clicks on any active back-end server, the admin sends Disable command to that back-end server which terminates it.

**Architecture Diagram and Interaction between components -**




- First, the client sends a request to the load balancer (super frontend server node) because this is the first point of contact for the client. The load balancer will have a list of active frontend servers and how occupied they are (here, occupied means the number of clients connected to them).
- The load balancer will find out the least occupied frontend server (the one who has least number of active connections) and redirects its request to that frontend server node. The load balancer also sends the list of active frontend servers to the admin console.
- Now the request goes to the frontend server and the frontend understands the request and figures out what it needs. Then it asks the master server which contains the root and metadata tablet about the whereabouts of the data it needs.
- Master server identifies which tablet would contain the user's information based on the first 2 characters of the username. From the tablet server mapping, it gets the list of all servers who contain information about the tablet. From this, the master server allocates one which has the least load and responds to the Front end server with the backend server's address.
- The Front end server directly talks with the backend server it received from the master server
- The master server collects heartbeat messages from the back end servers to identify which servers are active/inactive.

- The admin console asks for activity status of each front-end and back-end servers from load-balancer and master respectively and can give commands to terminate any active backend-server

**Overview of Features -**

Our Penn Cloud System supports the following features -

- Backend -
    - Distributed Key-Value store, synonymous to Google's Big Table structure. The Key-Value store is organized as a 'map' internally and is therefore, sparse. It supports 4 operations - GET, PUT, CPUT, DELETE.
    - Partitioning of storage data into different tablets, which are stored on different backend servers.
    - Replication of data across all backend servers in a server group.
    - Fault tolerant - Continues to serve requests as long as some of the replicas in the server group are alive.
    - Useful level of consistency is provided, even after the server crashes using logging and checkpointing. After the server comes back up, it is able to revert to its state before the crash.
- User accounts -
    - Allows users to sign up for new accounts and log in to created accounts. If an incorrect password is entered for an already existing user, he/she is not allowed to login.
    - On entering the correct password, the user is directed to a menu containing links to access Inbox and Drive.
- Webmail Service -
    - Users are able to access their inbox.
    - Users can open each email in their inbox to see details about the email - From, To, Subject and the content of the email.
    - Users can delete, reply or forward the email to another address.
    - Users can compose and send emails to other users within the system as well as to email addresses outside the system.
    - Accepts email from Thunderbird (outside the system).
    - Sends email to remote users outside the system.
- Storage Service -
    - Upload-Download web-storage system which allows users to upload files to the storage service and download files from their own storage.

    - Users can upload and delete files.
    - Users can create and delete folders (supports nested structure of directories).
    - Rename files and folders.
    - Move files and folders from one location to another within the drive.
    - Supports large files (both text and binary files).
- Admin console -
    - Shows the active frontend and backend servers
    - On clicking any active backend server, it will send a command to terminate the backend server

**Design Decisions -**

Backend -

- Tablet structure - The Tablet structure chosen was - 'aa_af', 'ag_az', etc.. and tablet IDs were assigned accordingly. Basically, on receiving commands from a particular user, we used the first 2 characters of the username to determine the tablet ID which would contain the user's information.
- K-V Structure - We created something similar to a tree based directory structure which has been illustrated above.
- Primary-based replication -
    - Read commands are handled by the individual backend servers.
    - All update commands - CPUT, PUT and DELETE are first forwarded to the Primary server of each server-group. The Primary server processes the command received and then propagates the command to all other servers in the group. The update command is then implemented across all replica servers.
- Consistency - We are following the Sequential consistency protocol. All write requests are sent out by the primary server to all other servers through TCP connections. This ensures that the writes are received in the same order.
- Fault Tolerance and Recovery - For ensuring recovery, we are maintaining one log file on each server and checkpoint files for each tablet on the server. These checkpoint files contain the key-value store of the particular tablet.
    - When a server crashes, it uses the checkpoint and log file in combination, to get back to its state before the crash. This ensures recovery.
    - As long as one of the replicas is available, requests will continue to be served.

Front-end -

- Since the backend only understands 4 commands - GET, PUT, CPUT, DELETE, the front end server uses these commands with appropriate parameters.
    - Create Account - PUT(<username>, pwd,<password>)
    - Login - GET(<username>,pwd)
    - Send email - PUT(<recipientUserName>,<email_timestamp>,<email content>)
    - Inbox - GET(<username>,emails)
    - Open particular email - GET(<username>,<email_timestamp>)
    - Delete email - DELETE(<username>,<email_timestamp>)
    - Forward email - PUT(<recipientUserName>,<email_timestamp>,<email content>)
    - Reply email - PUT(<recipientUserName>,<email_timestamp>,<email content>)
    - Upload a file on drive - PUT(<username>,<file absolute path>,<file content>)
    - Download a file - GET(<username>,<file absolute path>)
    - Move - CPUT(<username>,<old file absolute path>,<old file absolute path>,<new destination>)
    - Rename - CPUT(<username>,<old file absolute  path>,<old file absolute path>,<new destination>)
    - Delete on drive - DELETE(<username>,<file absolute path>
- Load Balancer - Automatically redirects the client's browser request to the front end server which has the least number of active connections.

Admin-console -

- The admin console server talks to load balancer, master server and backend server in the following way
    - Gets the front-end servers status from load-balancer
    - Gets the back-end servers status from master server
    - Sends Disable command to back-end server for termination

**Team Member Responsibilities -**

- Jay (Frontend) - Frontend server, Create account and Login, Email (Thunderbird, reply, forward, delete, SMTP Server, SMTP Client), Drive (upload, download), Load balancer

- Saurabh (Frontend) - Frontend server, Load balancer, Admin Console, Drive (upload, download, move, rename, delete)
- Shivani (Backend) - Replication, Logging & checkpointing, Heartbeats , Admin console
- Vandana (Backend) - Key-value store, Tablet memory management on Master server, Allocation of new primaries and replication synchronization