PennCloud Project Report

Team 16

Yiyan (Edger) Liang, Qintian Huang, Zining Liu, Enlin Gu

May 2025

Overview

This document presents the design, features, and major decisions for the PennCloud Project by Team 16.

---

1 Architecture Diagram

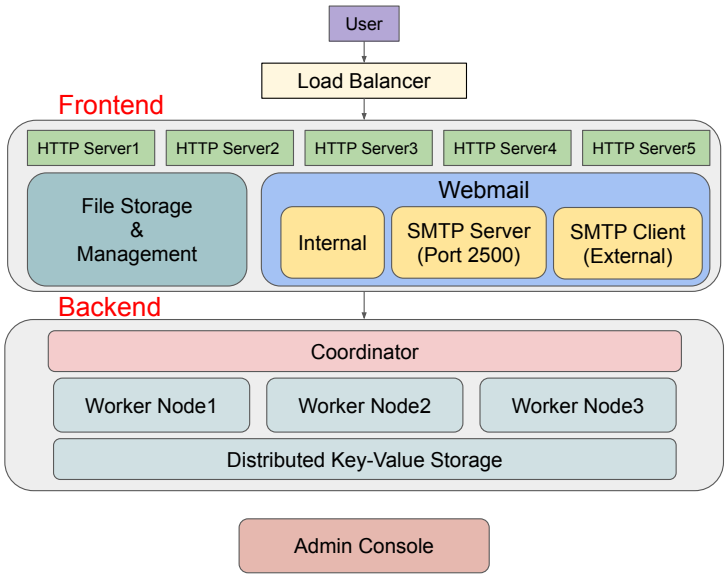

Figure 1: PennCloud provides users with webmail and storage services through a layered architecture. When users access the system, their requests first reach a load balancer that monitors server health and directs traffic to available frontend servers. These frontend servers handle the HTTP interface and user interactions, while delegating all data operations to the backend key-value store. The backend consists of a coordinator and multiple worker nodes that maintain replicated data with primary-backup consistency, ensuring fault tolerance through 3-way replication. System administrators can monitor and manage the entire infrastructure through an admin console, which displays server status and allows for manual intervention when necessary.

2 Frontend

2.1 HTTP Server

This part is implemented in `http_server.cc`.

- **User Interface**: We implemented a clear and easy-to-use user interface, including buttons to navigate among pages, auto refreshing and rendering content, and functional components, which will be discussed in the following sections.
- **Cookie**: It keeps the log-in status (both logged in and logged out) of each user and the cookie goes to the browser. When the user close the browser and re-open, or change a HTTP server, it will always keep the log-in status. The cookie information of each server goes into the KVS. Note that to make the load balancer and cookie fully functional, always using 127.0.0.1 instead of localhost when entering the address in the browser. This is because the load balancer always redirect to 127.0.0.1, and the cookies of 127.0.0.1 and of localhost are not the same one.

2.2 Load Balancer

This part is implemented in `load_balancer.cc`.

- The load balancer accepts incoming connections from the client at 127.0.0.1:8080, finds an available HTTP server, and redirects the client to the HTTP server by sending 307 Temporary Redirect. It will find the first available HTTP server. If there is no server available, it will send 503 Service Unavailable.
- The load balancer has a map of all HTTP servers and KVS servers, including their information: alias, status (ALIVE/DEAD), PID, IP, port. It sends these information to the admin console using JSON periodically to update the status of each server.
- The load balancer periodically tries to establish TCP connection with each HTTP server, while each HTTP server has a specific thread listening on the heartbeat thread, to keep track of the status of each HTTP server.
- The load balancer has ports to listen to HTTP servers and the KVS Coordinator for PIDs of each newly started server. It has direct connection to the KVS Coordinator, while the KVS Coordinator gets the PID from other KVS Workers.
- The load balancer also checks the status of each server by PID periodically and update the server status map, then let the admin console know which one is running and which one is down.

2.3 User Account

This part is implemented in `http_server.cc`.

- Our user account system can provide a complete authentication and user management solution for PennCloud. User can login with username/password credentials, and incorrect username/password will be rejected.
- The user can sign up for account. Once the account is created, Penncloud features are added to a user-based row key (row:`username`). The user can change password using account information management.
- When the user successfully login to the system, corresponding cookie is created and the user will be redirected to a main navigation menu for accessing PennCloud services.

2.4 Webmail System

This part is implemented in `webmail.cc, smtp_server.cc and smtp_client.cc`.

- **Complete Email Functionality:** The system provides a comprehensive web interface that allows users to manage their emails effectively. Users can browse their inbox with emails displayed in a sorted list, compose new messages with a user-friendly form, reply to received emails with the original message quoted, forward emails to other recipients while preserving the original content, and delete unwanted messages with immediate inbox updates.
- **SMTP Server:** The system includes a full SMTP server implementation that listens for incoming connections from external email clients. It handles the complete SMTP protocol flow including HELO/EHLO commands, sender and recipient validation, data transfer, and proper response codes. Upon receiving an email, it parses the headers and content, generates appropriate metadata, and stores the message in the distributed key-value store under the recipient's account for immediate access.
- **Dual-Strategy External Email Delivery:** The system implements two complementary approaches for external email delivery. For academic and corporate domains like seas.upenn.edu, it performs DNS MX record lookups to identify mail servers and communicates directly using the SMTP protocol with appropriate error handling and retry mechanisms. For major email providers like Gmailand Hotmail, the system automatically switches to a secure TLS relay approach with proper authentication credentials to maximize deliverability and avoid messages being flagged as spam.

2.5 File Storage System

This part is implemented in `http_server.cc`.

- Our file storage system can efficiently handle both small and large files with a flexible organization structure. The hierarchical organization can store files in folders and subfolders for easy organization. The user isolation mechanism keeps each user's files securely separated from others.
- Users can upload, download, rename and delete their files. They can also move files from one folder to another. Moreover, Users can create, delete and rename folders. They can also move folders to another folder. After these operation, files in one folder will remain inside.
- Users can upload and download both text and binary files. Binary files are encoded in BASE64 when uploaded, and are decoded when downloaded.
- Large files are divided into chunks when storing. There is a key for this file to track the size of the large file and the information of chunks.

2.6 Admin Console

This part is implemented in `http_server.cc` and can be accessed through `IP:Port/admin` or clickable links on the user interface.

- The admin console loads the configuration file of HTTP servers and KVS servers, and display them, including server alias, IP, port, status, PID, and buttons to **start** or **shutdown**. It handles both start and shut down in the admin console; start in terminal and shutdown in admin console; start in admin console and shutdown in terminal.
- When using the **start** button, it will start the HTTP/KVS server on the port shown by creating a new process. It also track the PID of the process and send it to the load balancer. When using the **shutdown** button, it kills the current process gracefully by PID.

- The admin console also shows the checkpoints of all KVS servers. It shows the current version number, users, and each user's files, and refreshes automatically.

3 Backend

3.1 KVS implementation

The backend is composed of two different functional parts, namely Coordinator and Server. The basic process for the frontend to interact with the backend is mainly like:

1. The coordinator should be accessed by the frontend first, and return the list of replica workers which is in charge of the tablet to the frontend for data editting.
2. Then, the frontend will send `PUT()`, `CPUT()`, `GET()` or `DELETE()` request to a corresponding worker, and the worker will execute the operation on the worker side.

3.2 Data Consistency

In the 3.1 part, the accessibility could be ensured, but the consistency between the replica workers should still need to ensured by a 2PC-like mechanism. In the project, the data consistency of writing (`PUT()`, `CPUT()` and `DELETE()`) within the same replica group is ensured by:

1. Upon receiving a write request (PUT, CPUT, or DELETE) from a client, the receiving node first communicates with coordinator to obtain an up-to-date list of all replica nodes responsible for the data item targeted by the client's request.
2. a. **If the current node is designated as the primary replica:** The primary node takes responsibility for coordinating the write operation. It sends the appropriate command (`REPLICA_DELETE`, `REPLICA_CPUT`, or `REPLICA_PUT`) to all other active worker nodes (secondary replicas) in the list.

   b. **If the current node is not the primary replica:** The node acts as a forwarder. It redirects the original client request directly to the node identified as the primary replica. At this point, the primary replica will then proceed as described in step 2a.
3. When other replica nodes receive a `REPLICA_DELETE`, `REPLICA_CPUT`, or `REPLICA_PUT` command from the primary replica:

   a. They first log the incoming request.

   b. An acknowledgment (ACK) message is sent back to the primary replica to confirm receipt of the command.

   c. Subsequently, each secondary worker begins executing the requested write operation on its local data copy.
4. The primary node executes the write operation on its own local data copy if received ACKs from all other replicas. In 2a situation, the primary node will directly send the response back to frontend; while in 2b situation, the replica node will first obtain the primary node's response and forward to the client.
5. The primary node executes the write operation on its own local data copy if received ACKs from all other replicas. In 2a situation, the primary node will directly send the response back to frontend; while in 2b situation, the replica node will first obtain the primary node's response and forward to the client.

6. If the primary cannot have all the ACKs from other replica workers, it will abort the operation and send back to all other workers, which will also abort the operation. The failure message will then be sent back to the frontend.

For the data reading operation (`GET()`), the consistency is not required, and the node will simply return the response to frontend. So, this protocal is basically equivalent to a Quoram Protocal with $W = N$ and $R = 1$.

3.3 Fault Tolerance and Recovery

Since 3.2 promised that the data will be consistent for each groups of workers handling the same tablets, to ensure the fault tolerance function, the checkpoint and logs are involved. This is composed of 3 parts:

- Worker liveness check: Coordinators could check the liveness of the workers, and update the available workers for each tablets. This is implemented by receiving the `HEARTBEAT` from the workers and update the liveness of each node in a periodical `liveness_check` thread inside the coordinator.
- Consistency checkpointing: The Coordinator send the checkpoint command to the same group of replicas periodically, and all the nodes made the checkpoints and clear the log at the same time.
- Consistent Recovery: Upon coming online, a worker node requests its replica list from the coordinator. Once received, if the worker identifies as primary (especially during an initial startup), it restores its state from a local checkpoint. If it's a secondary worker, it queries the primary for the current data version. The secondary then handles the primary's response: if the primary provides its version and relevant logs, the secondary compares this with its local version; if the primary doesn't respond (e.g., due to unavailability), the secondary attempts to update from its own local storage. If the versions are consistent or can be reconciled with the provided logs, the secondary executes necessary operations and appends the logs. However, if there's a version mismatch, the secondary defers to the primary, typically initiating a full state transfer or requesting more comprehensive logs to achieve synchronization.

4 Major Decisions and Challenges

1. Large request transmission between Backend and Frontend

   Challenge: While small data operations (e.g., short emails, small files) require only a single request to a worker node, large file operations ($\geq$ 10MB) involve multiple chunk transfers. During these extended transmissions, connections may fail, workers might become unavailable, or timeouts could occur.

   Decision: We implemented connection fault tolerance mechanism. Just like one-time operation, we first select an available worker. During operation, the status of the response is check to see whether the worker is still available. If not, it will shuffle worker list and pick another worker. An array is created to track the successfully operated chunks (eg. `CHUNK_RENAMED[]`), so we only need to process the unsuccessful chunks. Moreover, at global level, we set a flag to see if the chunked operation (put/get/del/rename) is successful. If not successful, it will retry this operation (`MAX_RETRIES = 3`).

2. Key value choice for easy storage management

   Challenge: Our file storage system needs to map hierarchical folder structures (with

chunked files) onto a flat key-value store while maintaining efficient access patterns and logical organization. Therefore, we should pick suitable key name.

Decision: We encode path hierarchy directly in the keys:
Row key: username
Column key: storage_data_full_path // For regular files
Column key: storage_meta_chunks_full_path // For chunked file metadata
Column key: storage_chunks_full_path_i // For individual file chunks
Column key: storage_meta_files // Comma-separated list of all file paths
Column key: storage_meta_folders // Comma-separated list of all folder paths

This approach allows our system to isolate users, display user file organization and easily access chunked files. It also supports folder and file upload/download/delete/rename and encode/decode/chunk operations.

3. Email Delivery Reliability to External Domains

Challenge: Dealing with various anti-spam measures implemented by receiving mail servers (like greeting delays and temporary rejections), handling different authentication requirements across mail providers, and ensuring proper email formatting to maximize deliverability.

Decision: We implemented a dual-strategy approach based on the recipient's domain. For standard SMTP servers, we use direct delivery with DNS MX record lookups and appropriate timing delays to handle anti-spam measures. For major email providers (Gmail, Hotmail, etc.), we implemented a secure TLS relay with proper authentication that significantly improves deliverability. The system automatically selects the appropriate strategy based on the recipient's domain and includes fallback mechanisms when the primary delivery method fails.