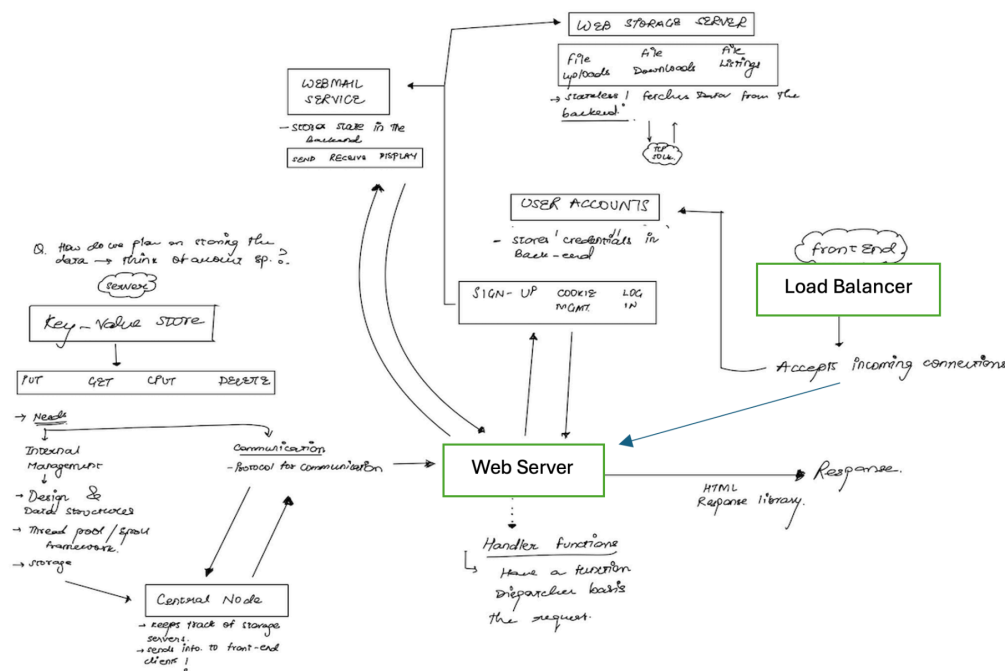


CIS 5050 PennCloud Final Report: Team T16

By Suchandra Chakraborty, Leo Gordon, Mayank Roy, Varun Singh



Part I: Design Architecture Description

Backend Architecture (Sub-part I)

The backend architecture of PennCloud is designed as a distributed, fault-tolerant system composed of four primary subsystems: the Key-Value (KV) Store, the Agentic (LLM) Service, the Webmail Server, and the Admin Console backend.

1. Key-Value Store & Distributed Consensus

The core of the backend is the Key-Value store, which serves as the source of truth for all persistent data.

- **Storage Engine:** To meet the requirement for $O(1)$ average-case time complexity, the in-memory data structure is an `unordered_map` (hash map). This ensures rapid data retrieval.
- **Consensus & Replication:** The system implements a highly read-available Quorum-based consensus and replication algorithm (**Paxos**). Write operations (**PUT**, **CPUT**, **DEL**) are broadcast to replica groups, requiring a quorum for the commit to succeed. This prevents split-brain scenarios and ensures data integrity.
- **Communication:** We use **gRPC** to handle requests, defined in the `server.protos` file. Additionally, legacy TCP ports are kept alive to support textual access.
- **Partitioning:** The system utilizes a partitioning strategy based on the key schema `<username>_<content_hash>_<type_flag>`. This allows for range-based partitioning where data specific to a user is co-located. To mitigate unbalanced loads (hot partitions), we implemented capabilities for hash-based partitioning as a fallback mechanism.
- **Processing:** It uses a threaded, worker queue event model to process requests efficiently.
- **Persistence:** A checkpointing mechanism writes the in-memory state to disk logs (`wal` files) to ensure data survives server crashes/restarts.

2. Agentic Service (LLM Infrastructure)

The Agentic Service is a specialized backend component designed to provide AI capabilities.

- **Model Hosting:** We host two (scalable up to N) Large Language Model (LLM) instances using `llama-server`. The specific model selected is `Qwen/Qwen2-1.5B-Instruct-GGUF`.
- **Process Management:** These processes are orchestrated via the `llmsetup.sh` executable, which handles the initialization and port binding of the underlying model servers.
- **Load Distribution:** The `llm_service.cpp` (and corresponding executable) acts as a proxy. It manages traffic to the model instances using a **Round-Robin** scheduling algorithm based on the total message number.
- **Interface:** The entire service has its own gRPC definition, located in `proto/llm_server.proto`.

3. Webmail Server (SMTP Infrastructure)

The mail system implements a complete SMTP server designed for both internal and external email delivery.

- **SMTP Server:** The server runs on a configurable port (default 2500), handling standard SMTP commands (`HELO/EHLO`, `MAIL FROM`, `RCPT TO`, `DATA`, `QUIT`).
- **Email Routing:**
 - **Intra-domain:** Local `@penncloud` emails are stored directly in the recipient's KV store row (`user:<username>:inbox`).
 - **External:** External emails trigger an MX record lookup via `res_query()` from `libresolv`, followed by SMTP relay to the destination server.
- **Compliance:** Outgoing relay messages include RFC 2822 compliant headers (From, To, Subject, Message-ID, Date) with CRLF line endings to ensure compatibility with external mail servers like Penn SEAS.
- **External Access:** The server accepts incoming mail from external SMTP clients like Thunderbird.

4. Admin Console Backend

The admin console acts as a monitoring aggregator.

- **Data Aggregation:** It queries all backend gRPC ports (9010-9017) to aggregate data across Paxos-replicated servers, deduplicating results with `std::set`.
- **Liveness Probes:** It uses a 1-second TCP timeout to detect health status, distinguishing between active and unresponsive nodes.

5. File Storage System and Collab Workspace

- **Path/Folder Manifest:** Each path and folder has a manifest that stores the name of files/folders. The KV stores the manifest as `<username + path, manifest, content>`.
- **Files:** Each file has a unique file ID stored in the KV, generated by hashing the username, path, and filename. Data is stored against the file ID in chunks.
- **Collab Workspace:** Files are stored against a global file ID to allow sharing. The collab space has its own manifest mapping usernames to files, controlling access. Every keystroke triggers a backend update for strict consistency.

Frontend Architecture (Sub-part II)

The frontend architecture is split into the HTTP Web Server (which acts as the Load Balancer and Logic Tier) and the Client-Side interfaces.

1. The Web Server & Load Balancer

The entry point to the system is a custom C++ HTTP server.

- **Architecture:** The frontend server is built as a multi-threaded system using conditional variables to handle clients. Each new connection is serviced by a thread.
- **Routing Logic:** The core routing logic is implemented in `htmlParser.cc`, specifically within the `prMap` method of the `HtmlParser` class. Whenever a GET or POST request is received, it passes through this map to invoke the appropriate handler (e.g., Inbox, Authentication, File Storage).

- **Smart Client / Load Balancing:** The frontend performs load balancing based on the number of clients on each server; the load balancer routes requests to the server with the least number of active clients. It performs health checks via a single HTTP request injection to each of the four servers.

2. Client-Side Interfaces

- **Standard Web Interface:** The primary UI is server-side rendered HTML/CSS and JavaScript.
 - **Webmail UI:** Routes such as `/inbox`, `/email`, `/compose`, `/webmail/sent`, and `/webmail/contacts` are registered in `htmlParser.cc`, with HTML generation handled in `email_html.h`. It features a consistent navigation bar across all views.
- **Collab Workspace (Extra Credit):** Built with JavaScript and WebSockets. It dynamically saves content typed by the user directly into the KV store.
- **Agentic Service UI (Extra Credit):** A simple front end implemented in JavaScript. It communicates asynchronously with the backend to allow for a streaming chat experience. (Future scope: persisting chats in the KV store for retrieval).
- **Admin Console UI:**
 - **Dashboard (`/admin`):** Shows server status with **ALIVE/DOWN** indicators.
 - **KV Data Viewer (`/admin/data`):** Displays a paginated, color-coded data table.
 - **Controls:** Provides Kill/Restart buttons for fault tolerance testing.

Part II: Overview of Features

Backend Features (Sub-part I)

1. **Key-Value Operations:** The backend supports atomic operations defined via gRPC:
 - **GET:** Retrieves value by key.
 - **PUT:** Inserts or overwrites data.
 - **CPUT (Conditional Put):** Performs an atomic compare-and-swap operation, essential for managing locks and concurrent edits in the collaboration workspace.
 - **DELETE:** Removes data permanently.
 - **GET_ROW:** A specialized feature allowing the retrieval of all columns associated with a row key (e.g., retrieving all emails for `user:varun:inbox`).
2. **Persistence & Recovery:** Includes a logging mechanism that records every write operation to disk. On startup, servers read this log to reconstruct the in-memory `unordered_map`, ensuring no data loss occurs during restarts.
3. **Admin Control Plane:** Implements a logic gate using atomic booleans. This allows an administrator to trigger a "Soft Shutdown" via RPC. In this state, the server remains alive to the Admin Console but rejects user traffic with an **UNAVAILABLE** status code, forcing the frontend to redirect traffic to healthy nodes.
4. **LLM Rotation:** Tracks the number of requests processed and rotates the active LLM port modulo $\$N\$$, ensuring even wear and thermal management on the hosting hardware.

Frontend Features (Sub-part II)

1. **Webmail Service:**
 - **Inbox/Sent Views:** Parses the backend KV store to render lists of emails with timestamps and subjects.
 - **Composition:** A rich-text compatible entry form allowing users to send messages to other `@penncloud` users and external addresses via SMTP relay.
 - **Reply/Forward:** Automatic quoting of original messages with pre-filled recipient and subject fields.
 - **External Mail:** Accepts incoming emails from SMTP clients like Thunderbird.

- **Sent Folder (Extra Credit):** Automatically saves outgoing emails to `user:<username>:sent`.
 - **Address Book (Extra Credit):** Contact management (add, view, delete) with one-click compose integration.
2. **File Storage Service:**
 - **Upload/Download:** Supports chunked binary uploads. Large files are split into 512KB chunks, encoded in Base64, stored across multiple KV keys, and reassembled upon download.
 - **File Management:** Users can rename files, move files between directories, and delete content.
 - **Virtual Directory System:** Implements a manifest-based directory system, allowing users to create nested folders without physical directory creation on the server disk.
 3. **Collaboration Workspace (Extra Credit):** A real-time document editor allowing multiple users to edit text simultaneously. It logs every key operation to the backend's KV structure for strict consistency and shows real-time user typing indicators.
 4. **Agentic Interface (Extra Credit):** A chat interface allowing users to query the hosted LLM. The frontend handles context and formats the prompt before sending it via gRPC to the `llm_service`.
 5. **Admin Console:** A dashboard visualizing the network topology. Displays health status (ALIVE/DOWN), current load, and provides control buttons (Kill, Restart, Soft-Shutdown). This service is visible only to the `admin` user.
-

Part III: Design Decisions & Major Challenges

Backend Design Decisions & Challenges

1. **Partitioning Strategy:**
 - *Decision:* Utilized a schema-based partitioning strategy where keys are prefixed with usernames.
 - *Reasoning:* Enables "Data Locality." All data for a specific user is stored sequentially or on the same shard, making operations like "Load Inbox" significantly faster than random distribution via pure hashing.
 - *Challenge:* "Hot Partitions" (one user having significantly more data). Mitigated by designing support for hash-based fallback if a single user's range exceeds a threshold.
2. **The "Soft Shutdown" Challenge:**
 - *Challenge:* Performing maintenance on a backend node without dropping active user requests or causing errors.
 - *Decision:* Implemented a "Control Plane" in gRPC. A global atomic boolean flag `is_active` guards all data operations. When set to false, the server returns a specific error code that the Frontend interprets as a signal to failover seamlessly.
3. **LLM Resource Constraints:**
 - *Challenge:* Hosting LLMs is memory-intensive and risks OOM crashes.
 - *Decision:* Selected `Qwen2-1.5B`, a quantized model fitting within 2GB of RAM. Constrained the context window size and used round-robin scheduling to ensure one instance processes a query while the other resets/cooling down.
4. **Email Storage Schema Design:**
 - *Decision:* Chose row-based storage pattern `user:<username>:inbox` with email IDs as columns.
 - *Reasoning:* Enables efficient per-user queries using `GetRow()` to fetch all emails at once. Data is serialized using a pipe-delimited format to avoid external libraries.
5. **Admin Console Aggregation:**

- *Challenge*: Initial implementation only queried 1 backend server, missing replicated data.
- *Decision*: Query all backend gRPC ports (read from `kv_servers.txt`), create separate connections, and merge results with deduplication via `std::set`.

Frontend Design Decisions & Challenges (Sub-part II)

1. Concurrency Model:

- *Decision*: Adopted an event-driven, thread-pool-based architecture rather than forking a new process for every request.
- *Reasoning*: Creating/destroying threads is computationally expensive. A fixed-size thread pool utilizing a synchronized work queue and conditional variables minimizes context switching overhead and prevents the server from being overwhelmed by a "thundering herd" of requests.

2. Client-Side Load Balancing (Smart Client):

- *Decision*: Moved routing logic into the Frontend C++ Web Server instead of using a central hardware load balancer.
- *Reasoning*: Allows the web server to maintain a health map of the backend. If a backend node is slow or paused, the frontend adapts instantly without waiting for a central authority update.

3. File Chunking & Base64 Encoding:

- *Challenge*: Storing large binary files in a text-based KV store is inefficient and error-prone due to delimiter collisions.
- *Decision*: Implemented a chunking mechanism where files are split into 512KB segments and Base64 encoded.
- *Trade-off*: Increases storage size by ~33% (Base64 overhead) but guarantees data integrity and bypasses the maximum value size limit of the KV store.

4. Command Pattern Routing:

- *Decision*: Utilized a `std::map` associating string paths (e.g., `/login`) with lambda functions in `htmlParser.cc`.
- *Reasoning*: Avoids massive `if-else` blocks, making the code modular. Adding features like Agentic service or Mail simply required adding one entry to the map.

5. Dynamic HTML Injection:

- *Challenge*: The Admin Console button needed to be visible only to the "admin" user.
- *Decision*: Refactored HTML generators to accept session context (username). Used C++ Raw String Literals (`R"(...)"`) with interpolation to dynamically inject the button server-side, ensuring security (button is not sent to non-admins).

6. Admin Port Architecture:

- *Decision*: Display `admin_port = base_port + 2000` in the UI while backend listens on base port.
- *Reasoning*: Designed for future implementation where admin traffic is isolated on a separate port for security. Currently, the port conversion is handled before connection.

Extra Credit Features

Webmail Extra Credit:

- **Sent Folder**: Saves a copy of outgoing emails to `user:<username>:sent`, viewable in a separate UI.
- **Address Book**: Add/view/delete operations for contacts stored as `name|email` pairs, with one-click compose integration.

Agentic Chat Service:

- A fully integrated LLM chat interface allowing real-time AI interaction.

Collab Workspace:

- A Google Docs-style application utilizing WebSockets for a TCP stream over the existing connection.
 - **Features:** Create/edit documents, share via username, owner deletion capabilities, and real-time cursor tracking to see where other users are typing.
-

Contribution

Frontend Development:

- **[Varun]:** Developed the Web server, HTML routing logic, Smart Client Load Balancer implementation, Session Management (Cookies), and User Authentication.
- **[Suchandra]:** Built the complete Webmail Interface (Inbox, email viewer, compose form, forward, reply, delete) and email routing logic.
- **[Varun/Leo]:** Implemented the File Storage system, including chunking algorithm, Base64 encoding, and virtual directory manifest system, and chunked transfer encoding.
- **[Suchandra/Mayank]:** Developed the Admin Console Dashboard with server status indicators and kill/restart controls.
- **[Suchandra]:** Implemented the KV Data Viewer for Admin Console with multi-shard aggregation, color-coded rows, and pagination.
- **[Leo]:** Developed the JavaScript integration/Front-End for the Agentic Chat service.
- **[Varun]:** Developed the collaborative document feature and websockets interface.
- **[Suchandra]:** Developed the Email Extra Credit features: sent folder, address book with contact management.

Backend Development:

- **[Leo/Mayank]:** Implemented the core Key-Value store `unordered_map` logic.
- **[Leo/Varun]:** Implemented the key gRPC service definitions, communication protos, and CMAKE build system.
- **[Mayank]:** Developed the Paxos consensus algorithm and replication logic.
- **[Leo/Mayank]:** Implemented the Persistence Manager (Checkpointing and WAL).
- **[Leo]:** Built the Agentic Service backend, including `llama-server` integration, `llmsetup.sh`, and the round-robin scheduler.
- **[Suchandra/ Varun]:** Implemented the SMTP server with full protocol support (HELO, MAIL FROM, RCPT TO, DATA, QUIT)
- **[Suchandra]:** External email relay with MX record lookup/RFC 2822 formatting.
- **[Suchandra]:** Designed the email storage schema.
- **[Suchandra]:** Implemented the Admin Console backend with multi-server gRPC queries and liveness probes.