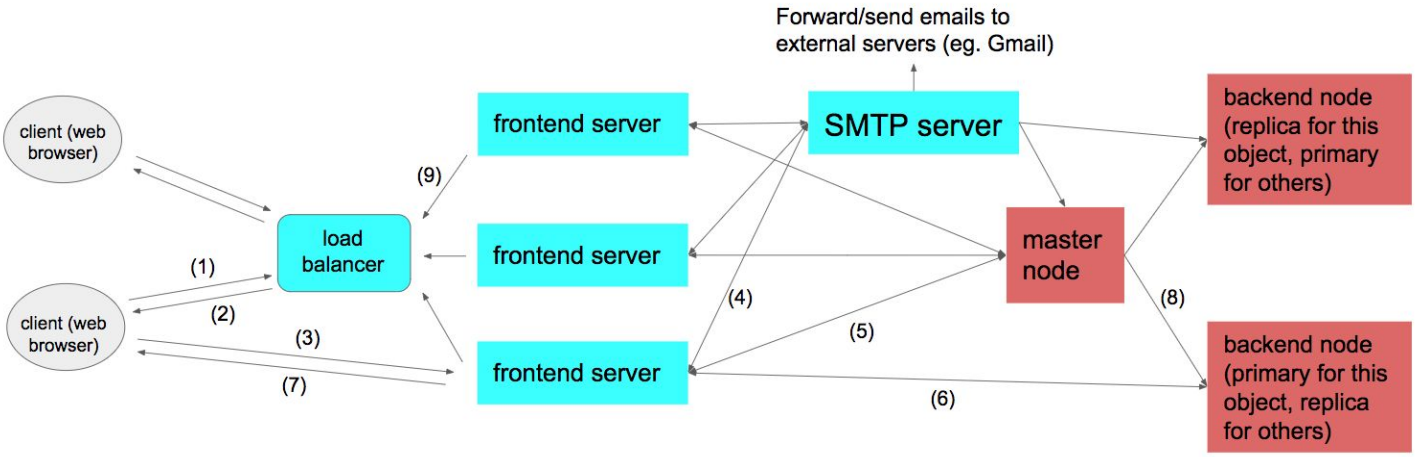
CIS 505 Final Project: PennCloud
Project Report

Design Description

- **Backend: Key-Value Store**
  - One master server (kv_master)
    - Provides way for frontend servers to learn the addresses of rows
    - Keeps track of which nodes are dead, alive, or recovering at any given time and provides coordination of backend servers during node failure and recovery
  - Many storage servers (kv_server)
    - Provides four basic operations (as well as admin endpoints) for frontend servers to interact with the distributed data store
    - Maintains consistency across data replications
    - Can recover from a node failure while maintaining consistency

- **Frontend: Client/Mail Servers**
  - All servers utilize an API to query the master backend node for appropriate primary and replica nodes for writing to and reading from tablets accordingly
    - The master node can also be queried to return a list of backend servers categorized by status (alive or dead)
    - Frontend servers can then query particular backend servers for key-value pairs stored in that server's tablets, ping the server for its latest status, and tell the server to shutdown or reboot
  - One load balancer server (load_balancer)
    - Find the minimal loaded server to redirect new connections across client servers
    - Keeps track of frontend server state by polling for a heartbeat from those servers (which are sent by active servers once every few seconds)
    - Robust against crashed frontend servers (only redirect to alive frontend servers)
  - One SMTP server (smtp_server)
    - All email operations go through the SMTP server (ie. the frontend servers send commands to the SMTP server to update and retrieve mailbox contents, and send emails to outside users---in some sense, this server also includes POP3 server features)
    - Communicates status of SMTP commands back to the frontend servers for the latter's error handling
    - SMTP client embedded into the server for purpose of sending email to outside users
  - Many frontend servers (frontend_server)
    - Each individually multithreaded, with one thread per HTTP request from the browser (ie. it's stateless)
    - Identify the user of each HTTP request via his/her cookie, performs storage updates by directly querying the backend and forwards email actions to the SMTP server
    - Allows logged-in users access to the admin console, which queries both the load balancer and master node (backend) for server status and tablet contents

Architectural Diagram



1) Initial request to load balancer
2) Load balancer response directing client to a specific frontend server
3) Client sends HTTP request to frontend server
4) If this is a mail request, the frontend server will forward necessary information to the SMTP server to handle said request
5) If this is a login or storage request the frontend server will ask the master node what the location of this user's row is
6) Upon receiving the row location, frontend server will handle the request with that backend node
7) Frontend server then sends an HTTP response back to the client
8) Master periodically sends heartbeats to all backend nodes to keep track of which nodes are alive and which ones need to be recovered
9) Frontend servers periodically send the load balancer a heartbeat with their current load statistic so the load balancer can choose the proper frontend server to forward the next incoming request to

Features

- **Backend: Distributed Key-Value Store**
  - Master node operations: GetLocation(row), GetServers(withStatus)
  - Storage node operations: PUT, GET, CPUT, DELETE as described in the project write-up
  - Storage node admin operations: ChangeState(to), ListTablets(), ListRows(tabletID)
  - Rows and columns are stored as strings with values as generic bytes
  - Supports a configurable number of:
    - Backend storage nodes (up to any arbitrary number)
    - Replication level (provides sufficient nodes exist)
    - Tablets on each storage node
    - Tablets to be held in memory at any given time
  - Handles concurrent reading and writing (all operations are synchronized at the row level via mutexes)
  - Replication: primary-based with a configurable number of replicas
  - Consistency: we offer sequential consistency for a given row, meaning all forwarding between primary and replicas, as well as checkpoint requests, for a given row are guaranteed to be considered in identical order across all nodes.

- Fault-tolerance: maintains a log of operations that is persisted to disk before any in memory changes to tablets are made
- Recovery: can recover if storage nodes fail at runtime (server can be primary for some objects and replicas for others) by synchronizing with the master and other relevant nodes for its assigned tablets

- **Frontend**
  - Backend API substantially augmented to give frontend servers “wrapper functions” to perform client-specific tasks (eg. ListMessages is an enhanced version of GET, DeleteFile is an enhanced version of DELETE)
    - Offers abstraction used by all frontend and SMTP servers
  - Load Balancer
    - Redirects new clients to particular frontend servers
    - Maintains status of frontend servers to ensure users are being serviced by an active frontend server
  - Accounts: login page (/login)
    - Sign up for a new account
    - Login as an existing user
    - Cookies: stores a cookie in the backend when a user logs in and uses this to verify user identity for future HTTP requests
  - Dashboard page (/dashboard)
    - Homepage for logged-in users: provides buttons/links to all other pages
    - Email page (/email)
      - Two column format
        - Sidebar shows a formatted list of emails with email subject/header, sender, arrival time, and preview (ie. first sentence or so of email)
        - When an email is clicked in the sidebar, the right column shows the contents of the email along with appropriate actions (see below)
      - Various email actions: compose new email, reply, delete, forward
      - Accepts emails from outside system (i.e. from SMTP client within VM and/or host machine)
      - Can send emails to remote users outside system (eg. Gmail, SEAS accounts)
  - Storage
    - Storage page (/drive)
      - Files and folders laid out in block layout, ordered by creation time
      - Provides the following functionality for files: upload, download, deletion, renaming, and moving from one directory to another
      - Provides the following functionality for directories: creation, renaming
      - Supports file sizes up to 10 MB (can support larger size by creating more space in memory) and arbitrary file types including text and binary files (images, pdfs...etc) through a base64 encoding scheme
  - Admin Console
    - Special webpage (url is http://localhost:$PORT_NUMBER/admin) accessible by logged-in users
    - View nodes in system (all servers, frontend and backend)
      - Current status display (alive or down --- for frontend servers this means load and last heartbeat timestamp)
      - Show raw data in backend storage as table of key-value pairs on a per server basis (clicking on a backend server shows the key-value pairs on that server)
    - Allows the user to stop/restart backend nodes to test fault-tolerance and recovery

Major Design Decisions/Challenges

- **Assumptions**
  - Backend master node does not fail (required to forward every request a client makes)
  - Frontend load balancer isn't down when a new user connects
  - SMTP server does not fail
- **Tablet grouping**
  - Tablets are defined by two numbers, tablet_group and tablet_num
  - For a backend cluster of *n* nodes, we have *n* tablet groups. The number of tablets within each group is configurable
  - A row is hashed twice to determine where it is stored: once to determine the tablet group and again to determine the tablet number within that group (using two separate hash functions)
  - Initially, every server is the primary for the tablet group with its server number (ie. server number 1 is primary for tablet group 1), we then have some number of replicas for each tablet group
  - When a node needs to recover, all synchronization steps are done for each tablet group it is responsible for (dictated by master) and those steps could be different depending on whether that node is a primary or replica for that tablet group (again dictated by master)
  - We created the notion of a 'tablet group' in response to the fact that primaries can change after nodes fail and recover
- **Replication: primary-based**
  - Reads: backend master node tells frontend servers that they can read (GET) from any replica
  - Writes
    - Master tells frontend servers that they can write (PUT, CPUT, DELETE) to the primary for that object
    - Primary uses synchronous forwarding of all messages to replicas: wait for ACK or that the replica has failed from every replica before forwarding the next message
  - Assumption: number of backend servers must be at least NUM_REPLICAS + 1 (one server to serve as the primary, and then NUM_REPLICAS replica servers): enforced by preventing master from starting up if this is not the case
- **Fault-tolerance and recovery**
  - Logging: maintain one log per tablet on each server. We log all API calls that modify a tablet before that change is written to the in memory tablet. When a tablet is evicted from the cache and a checkpoint is initiated, this log file is cleared.
  - Checkpointing: centralized, so replicas only checkpoint when the primary tells them to. All nodes maintain a counter which is then used in the tablet filename to aid in recovery synchronization.
- **Consistency**
  - Write requests sent to a primary are forwarded to each of the servers that are replicating that tablet group. This is done while the row lock for the original write is held to ensure that forwards of writes to the same object are received in identical order
  - It is possible that writes will occur in a different order across rows; however, because the destinations are different this will never result in divergence between the primary and replicas.
- **Separation of frontend and backend**
  - Backend fundamentally only implements the four calls from the writeup
  - Frontend augments the backend's 4 fundamental calls to provide frontend servers ease of access into the backend servers. Originally, these functions were implemented in the backend but were moved these to the frontend on Prof. Linh Phan's advice. This minimizes complexity with regards to backend goals (replication, fault-tolerance, recovery)
- **Frontend server to SMTP server communication (and vice versa)**
  - All email operations go through the SMTP server, in effect making it a SMTP/POP3 hybrid
  - Frontend servers uses standard SMTP protocol (augmented to allow POP3-style commands) to send commands to SMTP server, which responds back via SMTP response codes

    - Ensure all emails have a complete header and subject to minimize chance of rejection from external servers (say, if a user was sending/forwarding an email to Gmail)
- **Storage service robustness**
    - Encodes all file contents into base64 to support storage of non-text files
    - Processes large files chunk by chunk when uploading and downloading
- **Frontend UI**
    - Frontend servers fetch appropriate html templates and then sends them to the client
        - Non-static information (such as server responses) are sent to the client separately as JSON-style strings
    - HTML templates also contain necessary CSS and JavaScript to render the webpage
        - JavaScript used to parse the JSON objects received from the frontend servers and send appropriate asynchronous HTTP requests (ie. AJAX) based on user action (eg. pressing a button)
        - JavaScript used to store state in the browser for each user for purposes of user interactivity (eg. setting different properties on HTML elements based on user action)

Team Member Responsibilities

- **Christopher Fischer: Backend**
    - Key-value API (4 basic; admin; heartbeats) and test bench
    - Associated API “wrapper functions” for frontend servers
    - Replication: logging, checkpointing
    - Tablet memory management
    - Key-value master server
- **Leon Wu: Backend**
    - Replication: message forwarding
    - Recovery: synchronization, assigning of new primaries
- **Wei Zhang: Frontend**
    - Webmail service & webpage, including SMTP server/client and HTTP-to-SMTP pipeline/integration
    - Admin console (+ associated HTML/CSS/JavaScript)
- **Shi Shu: Frontend**
    - Load balancer, frontend server (parsing http requests and reply results to browsers)
    - Login, Dashboard, Storage service (+ associated HTML/CSS/JavaScript)