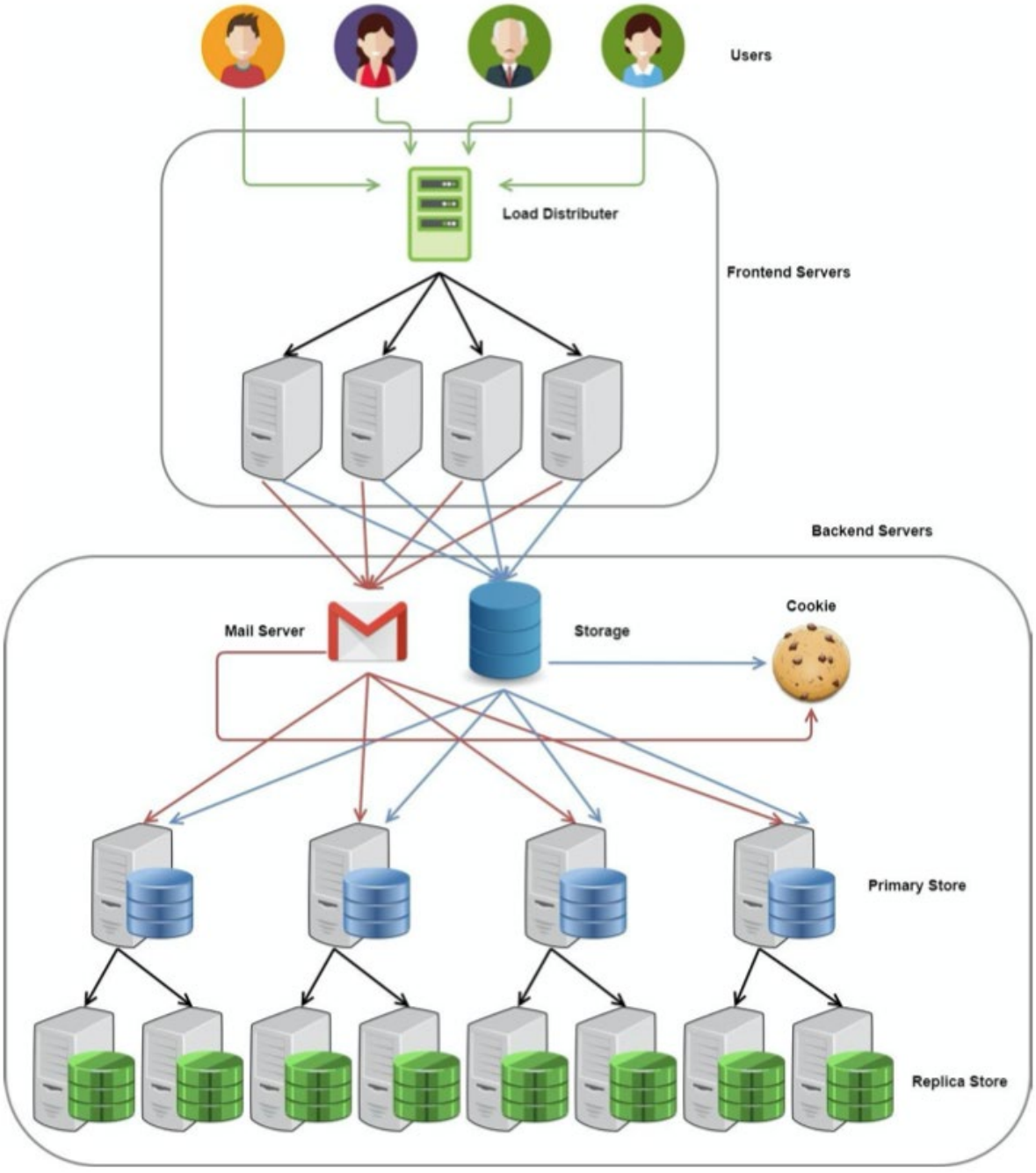
CIS505 Project Report - PennCloud

1. Design



2. Overview

(1)Frontend

a. Web interfaces

For the web interfaces, we designed several predefined HTML templates for different views and the format for request and response. When we receive a http request from the user, our frontend will parse it and send it to our backend service for further processing. Then, the backend service will write the result back to the front end. The frontend will then parse the response from backend service, pick the correct HTML template and populate it with the parsed information.

For the communication between our frontend and backend, we are using string based transmission. All the message payload, especially those containing non-ASCII characters such as the binary string of a photo or file, were encoded/decoded to base64 formatted and processed as a string.

b. Email service

For each of the users, they will have 2 email lists to keep track of all the emails, 1 list for the inbox and another for the outbox. To avoid collision, all the mails will be assigned a MD5 hashcode to ensure its uniqueness. This hashcode is also of great importance during the retrieval from inbox. For example, if we need to retrieve a specific email from the bigtable, the front end server will send a GET operation to the backend service with the username, required source and hashcode. By doing so, we can guarantee that all of the mails were correctly saved and will not collide with each other.

For the web mail service, apart from sending mails to the localhost, it can also send mails to the outside world following the SMTP protocol. One of the frontend servers will parse all the information including the sender, recipient, content and etc. Then, it will call a SMTP client to communicate with the SMTP server. Similarly, when a SMTP server receives an inbound email, it will also generate a hash value, attach it to the mail and call the PUT operation to store it in the backend. Besides, users can also extract a vector of emails from the backend service and render it within the browser for display. Update operation will also take the username, email address and mail type(inbox or outbox).

Regarding the consistency model, we are using client-centric consistency here since it is more intuitive and suitable for our use case. Each user can log into their own account and we only need to manage what a specific user should be able to see. All the mails were replicated throughout different servers and assigned to each of the users. We are also using “Read your writes” here since we need to ensure the sequence of operation that a user can see.

c. Load balancing

To ensure the robustness and capacity of our service, users are totally blind to the real address of all the frontend servers. There is a special URL(default to be localhost:8090) that was exposed to the users. The users only need to access this URL and the load balancer will automatically route all the user requests to one/several specific frontend servers. By doing so, during the peak hour, we can easily enlarge the throughput by launching more frontend servers to handle incoming requests.

To balance the incoming requests equally and monitor the condition of each server, we are using round-robin algorithms. Within the frontend server, we maintained a doubly-linked queue and pushed all the servers into it. Whenever we have a new incoming request, the server at the front of the deque will handle the request and move to the end of the queue. During the initialization of load balancer, a child thread will be initialized to check the state of each server every several seconds. If the server is done, it will also be moved to the end of the queue. If all the servers are down, the message will be no longer available and the users will receive a message.

(2)Backend

For the backend service, we had implemented a series of features including upload/delete files, create/delete folders, rename/move files/folders and etc. In our implementation, since we don't need to consider disconnected operation, we can use primary-based protocols with remote writes. It is also much easier to implement when comparing with the quorum-based protocols. All the nodes in our system can send read/write messages to other nodes and one primary node will coordinate the behavior across all the nodes.

a. Communication

The nodes are talking with each other with some predefined rules. All operations between primary nodes and replicas will follow the same template. The message starts with the type of its payload, showing whether it is for checkpoint, heartbeat, election or the user related operations such as PUT/CPUT/GET/DELETE. Then, the message payload was separated with the delimiters for different kinds of information. The receiver will parse the inbound message and respond to the sender based on the same rules.

b. Storage

Under the hood of the storage implementation, there is a LRU cache that stores everything. Each node will maintain a tablet of the bigtable, and all the tablets are distributed evenly on different groups. For each user within the tablet, they were assigned to an LRU cache. The LRU cache will keep monitoring the memory usage and flush the unnecessary data to the disk, or read necessary data from the hard disk into our memory.

3.Design and discussion

(1)Message ordering

For the write operations, it is also important to decide the correct ordering since the operation between different users can come across. Fortunately, with the nature of the system we are building, it is enough for us to maintain a FIFO ordering only instead of a total ordering. The primary will deliver the message to application (bigtable) first and send the messages to all the replicas and the replicas will deliver it accordingly. Since all the messages are user specific and the operation of user A should not interfere with the operation of user B. So, we only need to maintain the relative sequence of operations based on each user, instead of all the operations.

(2)Replication protocols

To preserve the simplicity, we are using primary-based protocols instead of the quorum-based one. In our implementation, since we don't need to consider disconnected operation, we can use primary-based protocols with remote writes. All the nodes in our system can send read/write messages to other nodes and one primary node will coordinate the behavior across all the nodes.

Only the primary node can accept write operations(PUT/CPUT/DELETE) and all the nodes can accept GET operations. For the primary node, when it receives a write operation, apart from multicasting it to all the replicas, a sequence number will be generated first and appended to the front of the message. By doing so, the message routed to all the replicas will contain both sequence number and message payload, which can guarantee the virtual synchrony. Once the operation succeeds, the replicas will reply a SUCCESS message back to the primary node.

(3)Recovery

Each data center can keep a log or the operations received for failure recovery. The messages that may interfere with the state of the storage will be force-append at the tail of the local log. It also keeps a periodic checkpoint with a version number on the local disk.

When a node needs to recover from failure, it will contact the master server first to figure out the primary node of the current group. Then, it sends the largest sequence number within the log file to the primary node. Based on the response from the primary node, the recovered node will replay its log or request some extra log from the primary node.

(4)Heartbeat

The primary node of each group will have a separate thread to send heartbeat messages to the master node every several seconds. Thus, the master node can monitor the status of each of the primary nodes. When any of them fails, it will contact the replicas and promote a new primary node for that group. The decision will also be broadcasted to the whole group.

4.Division of labor

Frontend server: Kai Jin, Haotian Zhu
Backend server: Jin Zhao, Yankai Liu