COMPSCI 3012 - Distributed Systems

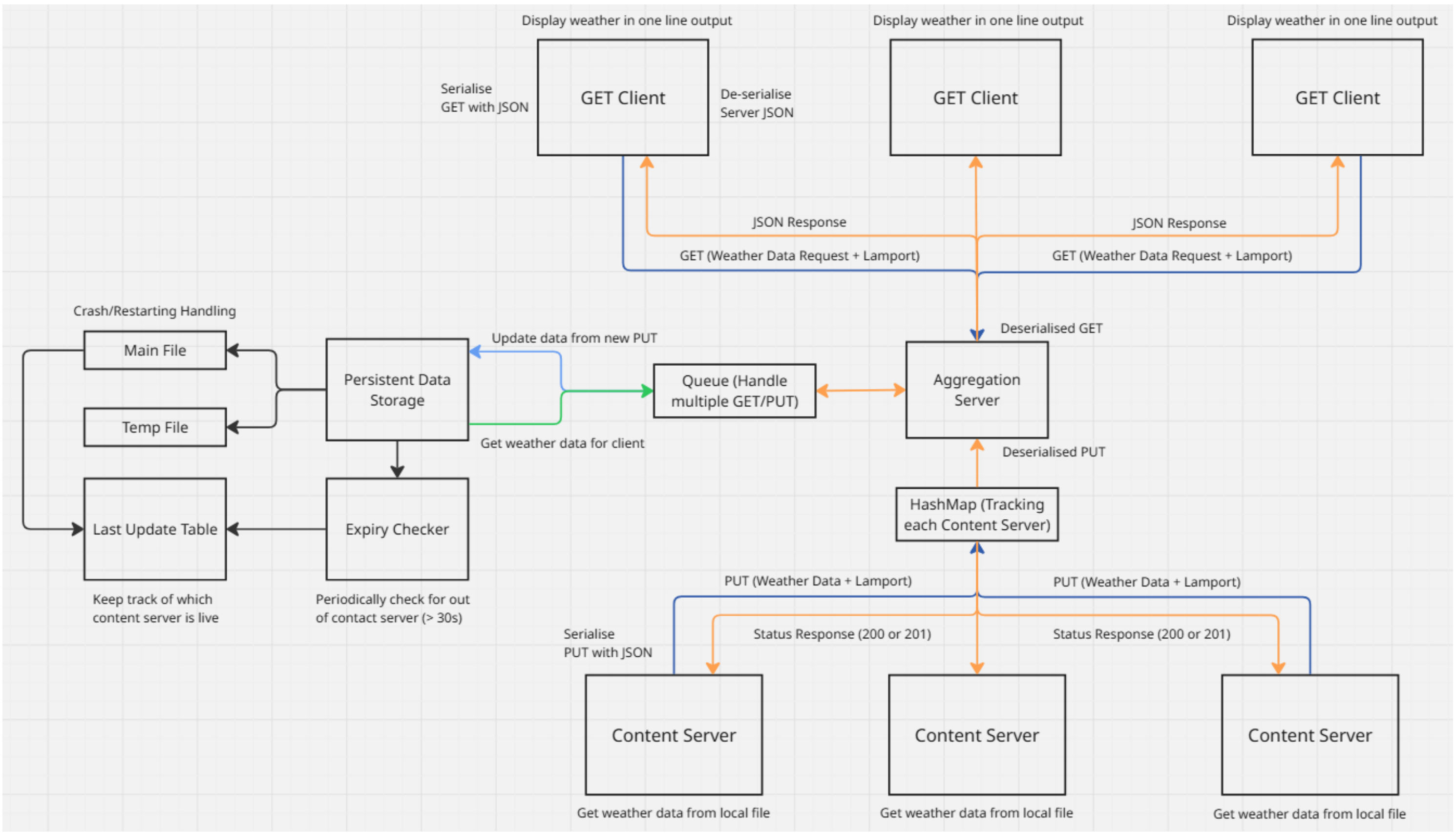
Gia Bao Au - a1897967

Assignment 2

Part 1: Design Sketch

For this assignment, we are building an aggregation server that accepts PUTs from multiple content servers and sends back weather data to clients in JSON format. We need to keep in mind of Lamport ordering, race conditions or deadlocks.

1. High-level System Architecture


Display weather in one line output
Display weather in one line output
Display weather in one line output
Serialise GET with JSON
GET Client
De-serialise Server JSON
GET Client
GET Client
JSON Response
JSON Response
GET (Weather Data Request + Lamport)
GET (Weather Data Request + Lamport)
Crash/Restarting Handling
Main File
Temp File
Persistent Data Storage
Update data from new PUT
Get weather data for client
Queue (Handle multiple GET/PUT)
Deserialised GET
Aggregation Server
Deserialised PUT
HashMap (Tracking each Content Server)
Last Update Table
Expiry Checker
Keep track of which content server is live
Periodically check for out of contact server (> 30s)
PUT (Weather Data + Lamport)
PUT (Weather Data + Lamport)
Serialise PUT with JSON
Status Response (200 or 201)
Status Response (200 or 201)
Content Server
Content Server
Content Server
Get weather data from local file
Get weather data from local file
Get weather data from local file

2. System Components

AggregationServer

Handling request (PUT + GET):

- **GET handler**: reads from persistent data storage and writes JSON response for sending back to client.
- **PUT handler**: accept PUT requests from content servers and check if we need to update the server memory.
- **Request parser:** extracts JSON body and header from client or content server request.
- **Request queue**: use a priority queue for pending PUT/GET, which can be ordered by Lamport Timestamp (Arrival Sequence for tiebreaker following FIFO)

Locating + Update data:

- **Update applier**: applying updates to in-memory storage and persistent file.
- **Crash-safe updates** : using atomic writes and write-ahead log (WAL) for rejecting an update when a crash happens.
- **Expiry checker**: remove stale data of content servers, which are out of contact for more than 30 seconds.

Content Server

- Read weather data from local files and convert them to JSON format.
- Send this data to aggregation server as PUT request.
- Manage and update its Lamport clock before and after the server response.

GET Client

- Retrieve weather data from aggregation server using GET.
- Parse and display the JSON data in one line output.
- Manage and update its Lamport clock before and after the server response.

3. Use Cases

Use case 1: Uploading Weather Data

1. Content Server reads from local file for new data.
2. Increment its own Lamport Clock before sending
3. It converts the data to JSON and sends as PUT request to Aggregation Server
4. Aggregation Server receives the PUT request and check if its valid
   - Return 201 for new connection
   - Return 200 for successful upload

Use case 2: Client retrieving weather data

1. Client sends a GET request for weather data, which could be the latest or after a specific Lamport Timestamp
2. Aggregation Server retrieves the request and reads from the Persistent Data Storage. Depending on the requirement from the client, the server prepare the weather data to send back.
3. The aggregation server sends back weather data in JSON format
4. The client parsed it and display in one-line output.

Use case 3: updating an existing record on the Aggregation Server

1. The Content Server send PUT(payload_id = CS1, Lamport = 6)
2. The Aggregation Server accept this PUT (Send back 200 or 201) and check that it does have a record with payload_id = CS1 and a Lamport of 3
3. Since 6 is larger than 3, this means that the PUT request contains new weather data
4. Client who requests right after before this record expire will get the new record with Lamport 6

4. Replica strategy (how many servers?)

For this assignment, we will be using a single aggregation server as replication adds complexity such as consensus.

5. Testing plan

Unit tests

- Lamport Clock
- Testing for WAL and Atomic write.
- Apply update with different Lamport Timestamp

Integration tests

- Testing with multiple content servers:
    - Send multiple PUT request from content servers
    - Verify if the PUT came in order and appropriate 201, 200 code was returned
- Test GET from client to Aggregation Server: sending multiple GET request for weather data and check if the responses are consistent.
- Send PUT1 -> GET1 -> PUT2: GET1 should see the weather record from PUT1 as PUT1 comes first then PUT2 later.
- Send PUT1(Lamport = 4) -> PUT2 (Lamport = 8) -> GET1(Lamport = 6): GET1 again should see the server state after PUT1 is applied due to the PriorityBlockingQueue sort request by Lamport timestamp.