Design

A1755144 Cheng En Hsieh

Components:

1. Client
2. Aggregation server
3. Content server

Use cases:

- Client:

  Send a GET request to the aggregation server and display the replied ATOM feed

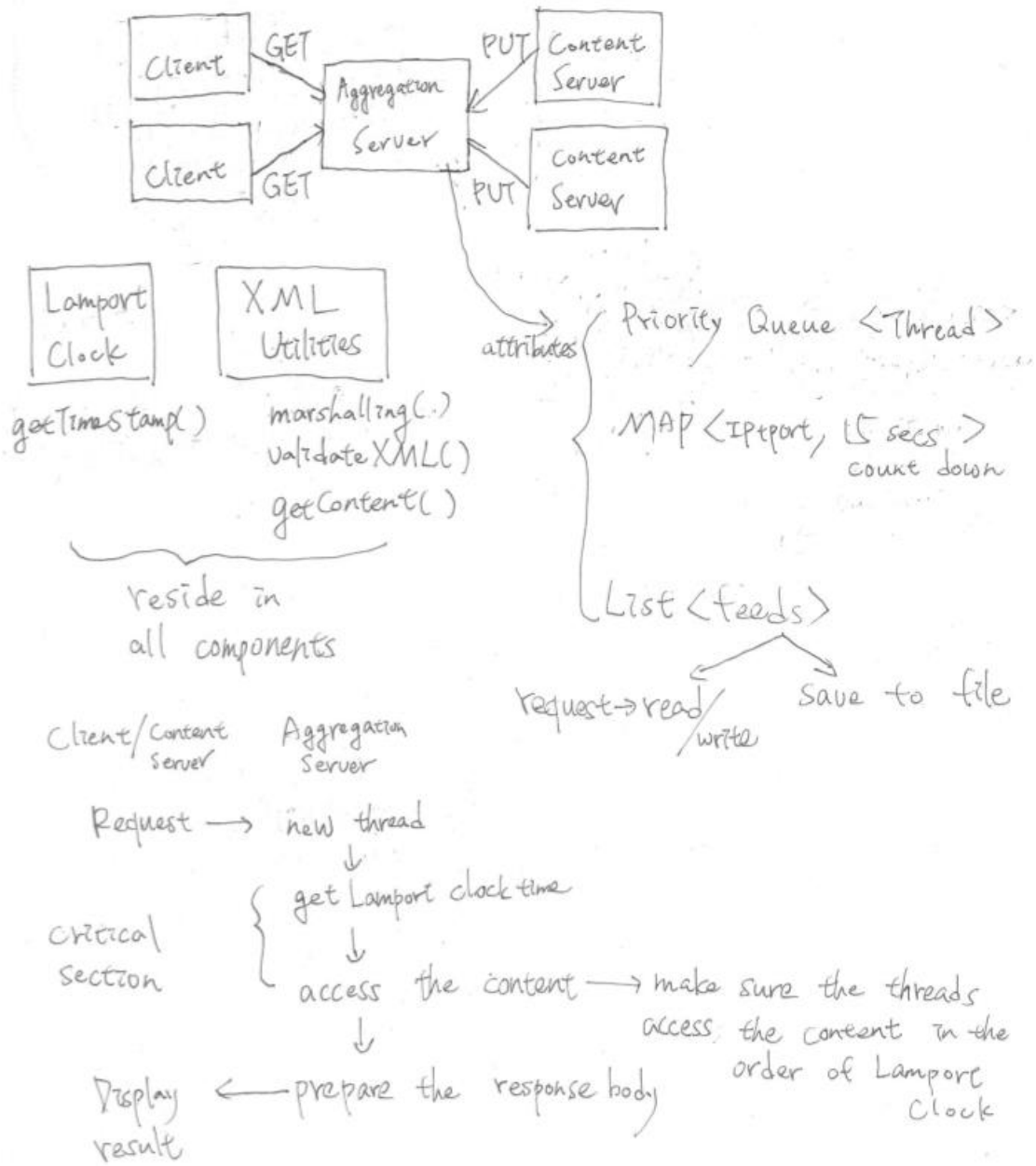
- Aggregation server:
  1. Receive a GET request from the client and return with an ATOM feed
  2. Receive a PUT request from the content server and update the content the aggregation server maintained
- Content server:

  Send a PUT request to the aggregation server, the content is loaded from a local file

System Requirements:

1. Aggregation server should aggregate the feeds send by content servers.
2. Aggregation server should response the clients with the latest feeds.
3. Content servers should be able to put their feed to the aggregation server.
4. Clients should be able to ask the latest feeds from the aggregation server.
5. There could be multiple content servers.
6. There could be multiple clients.
7. The aggregation server should delete the feeds of content servers that are no longer in contact in 15 seconds.
8. The aggregation server should response 500 when feed format error.
9. The aggregation server should response 400 when the request method is neither GET nor PUT.
10. The aggregation server should response 204 when the feed is empty.
11. The aggregation server should response 201 when it creates a feed.
12. The aggregation server should response 200 when it updates a feed.
13. The aggregation server should be fault-tolerant, it should be able to save the state of all the current feeds.
14. The aggregation server should be able to tell it is the same content server when a content server recovers from a crash.
15. The client should be fault-tolerant, it should retry after a connection failure.
16. The system should use a Lamport clock to sequence the events.

System Architecture:



Each Lamport clock maintains an event sequence number and changes it when an event happens.

The XML utilities can parse the content in XML format, a feed is presented in an XML format.

The Lamport clock and XML utilities reside in all the components of the system, so the time of the components in contact will finally converge and all the components can parse the content of feeds.

The use of “thread” in the aggregation server allows it to process multiple requests from multiple content servers and multiple clients.

The “map” in the aggregation server is used to maintain the timers for each feed. If a timeout happens, it will delete its’ corresponding feed.

The priority queue makes sure that the responses happen in sequence.

There is a list keeps all the feeds inside. Each feed is saved as a feed object which implements the Serializable interface, so the whole list can save as a file. Thus, the aggregation server can restore all the feeds after a crash.

System Requirements Verification:

Requirement Number	Test Case
1,2,3,4	Start an aggregation server, use a content server to send a put, and see if a client can successfully get the feed.
5	Use multiple content servers to put feeds on the aggregation server.
6	Use multiple clients to get the feeds from the aggregation server.
7	Put a feed, wait for 15 seconds, see if the feed still exists.
8	Send a bad feed and see the response.
9	Send a request that is neither PUT nor GET.
10	Send an empty feed.
11,12	Send multiple PUTs in a row from a single content server.
13	Shutdown the aggregation server and bring it up again and see if the feeds still exist.
14	Send one PUT, shutdown the content server, bring it up and send a PUT again, see if the aggregation can successfully tell they are from the same content server
15	Start a client first and then start the aggregation server, see if the client can successfully get the correct message from the aggregation server.
16	Send multiple PUTs from different content servers. See if the client can get the feeds in the right sequence.