CSCI/ECEN 5673: Distributed Systems
Spring 2023

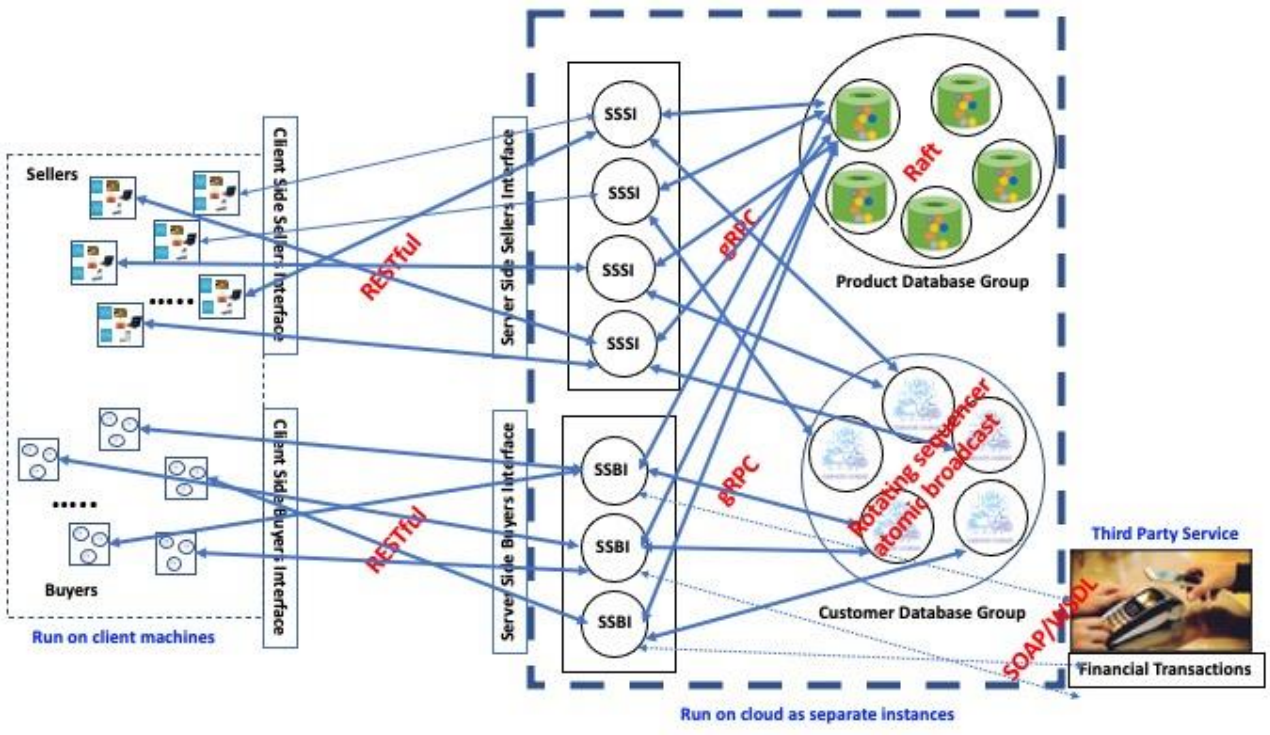
Programming Assignment Four

Due Date and Time: 11:59 PM, Tuesday, April 11, 2023

Goal: The goal of this programming assignment is to extend the online marketplace system you have developed so far in Programming Assignment One and Two as follows:

- Design and implement a rotating sequencer atomic broadcast protocol
- Replicate the customers database over five servers using your rotating sequencer atomic broadcast protocol
- Replicate the products database over five servers using Raft
- Replicate the server-side sellers interface over four servers
- Replicate the server-side buyers interface over four servers

You may work in teams of size two students



Characteristics of an item put up for sale, seller characteristics, buyer characteristics and the APIs of the logical components are same as in Programming assignment two. Also, the communication mechanisms between client and frontend (REST), frontend and backend (gRPC), and frontend and third-party service (SOAP/WSDL) are same as in programming assignment two.

A key limitation of the online marketplace system we have built so far is that the server (frontend and backend) is both a performance bottleneck and a failure bottleneck. In this assignment, we will address this limitation by replicating all logical components of server frontend and backend over different hosts.

CSCI/ECEN 5673: Distributed Systems
Spring 2023

Design and implement a rotating sequencer atomic broadcast protocol

Here is a high-level description of this protocol. You will first work out the details of this protocol and then implement it. Assume that a group has *n* members, each having a unique id, *0* ... *n-1*. A client may submit a request to any one of the group members. On receiving a request from a client, a group member sends a *Request message* to every group member. This message includes a unique request id <*sender id, local seq number*>, the client request and some additional meta data.

For each Request message, one of the group members assigns a global sequence number to this Request message and sends out a *Sequence message* to all group members. This message includes the assigned global sequence number, message id of the Request message and some additional meta data. The global sequence numbers are assigned in a monotonically increasing order starting from *0*. The global sequence number assigned to a Request message determines the delivery order of that Request message.

The task of assigning global sequence numbers and sending Sequence messages is shared by all group members as follows: A Sequence message with global sequence number *k* is sent out by the group member whose member id is *k mod n*. This group member assigns this global sequence number to a Request message with request message id <*sid*, *seq#*> and sends out a Sequence message after (1) it has received all Sequence messages with global sequence numbers less than *k* as well as all corresponding Request messages to which those global sequence numbers are assigned, and (2) all request messages sent by the member with member id *sid* and local seq numbers less than *seq#* have been assigned a global sequence number.

A group member delivers a Request message with assigned global sequence number *s* only after (1) it has delivered all Request messages with assigned global sequence numbers less than *s*, and (2) it has ensured that a majority of the group members have received all Request messages as well as their corresponding Sequences messages with global sequence numbers *s* or less. Use meta data in the Request and Sequence messages to determine these delivery conditions.

Group members use negative acknowledgement technique to recover from message losses. They send a *Retransmit request message* whenever they detect a missing Request message or a Sequence message. The Retransmit request message is sent to the sender of the missing message.

For this protocol, we will assume that the group members do not fail and they do not get partitioned from one another. So, there is no need for a group membership protocol.

Finally, group members use UDP as the underlying communication protocol for all communication.

Replication of server-side sellers interface and server-side buyers interface

Replicate server-side sellers interface and server-side buyers interface on four servers each. Since these components are stateless, there is no need for any protocol to manage these replications. Configure your client-side sellers interface and client-side buyers interface with a list of all replicas and let the client connect to one of them. In case a client gets disconnected from a replica it was connected to, it would try to connect to another replica from the list. In case one of the replicas suffers crash failure, just restart it (ensure that it has the same IP address).

Replication of customer database

Replicate customer database over five servers using your rotating sequencer atomic broadcast protocol. Assume that these servers do not fail, however communication is unreliable.

CSCI/ECEN 5673: Distributed Systems
Spring 2023

Replication of product database

Replicate product database over five servers using Raft. Download an implementation of Raft from the Internet. There are several open-source implementations available in different languages (See https://raft.github.io/). Assume that the servers can suffer crash failures and communication is unreliable.

Requirements of programming assignment four

Evaluation

Run your server components, including all replicas as separate instances on cloud. Report the following performance numbers:

- Average response time for each client function when all replicas run normally (no failures).
- Average response time for each client function when one server-side sellers interface replica and one server-side buyers interface to which some of the clients are connected fail.
- Average response time for each client function when one product database replica (not the leader) fails.
- Average response time for each client function when the product database replica acting as leader fails.

What to submit

Submit a single zipfile that contains all source code files, makefiles, and a README file. In the README file, provide a brief description of your system design along with any assumptions (8-10 lines), current state of your system (what works and what not), and all performance numbers.