CSCI/ECEN 5673: Distributed Systems
Spring 2020

Programming Assignment 2

Due Dates: 03/06/2020 (Phase I)
03/20/2020 (Phase II)

You may work in teams of two students to complete this assignment.

The goal of this assignment is to implement a fault-tolerant queue data structure called *FTQueue* that exports the following operations to the clients:

int *qCreate* (int *label*); //If there exists a queue associated with *label*, return the
//corresponding queue id; queue ids are unique positive integers
//Otherwise, create a new queue of integers associated with
//*label* and return a queue id
int qDestroy(int *queue_id*); //deletes a queue
int *qId* (int *label*); //returns queue id of the queue associated with *label* if one exists
//Otherwise, return -1
void *qPush* (int *queue_id*, int *item*); // enters *item* in the queue
int *qPop* (int *queue_id*); // removes an item from the queue and returns it
int *qTop* (int *queue_id*); // returns the value of the first element in the queue
int *qSize* (int *queue_id*); // returns the number of items in the queue

FTQueue is replicated over *n* servers and must be able to tolerate server crash failures as well as communication omission failure including network partitions.

You will implement *FTQueue* in two phases. The first phase is due on March 06 and the second phase is due on March 20.

Phase 1

In this phase, implement *FTQueue* assuming that there are no server failures or network partitions. However, the communication system may suffer from omission failures, which means messages may get lost. In particular, use UDP as your underlying communication protocol and negative acknowledgement technique to recover from message losses.

To ensure the consistency of your replicated data structure, implement a group communication middleware that includes a total order, reliable atomic multicast protocol as described below:

To multicast a message, a group member sends that message to every group member. Assume that the group members have unique ids, 0 to n-1. For each multicast message, one group member sends out a global sequence for that message to all group members. This global sequence number determines the delivery order of that message. Global sequence number *k* is sent out by group member *k mod n*. Group members deliver messages in the order determined by their global sequence numbers.

Phase II

Extend your group communication middleware from phase I to tolerate server failures and network partitions via a group membership protocol that implements extended virtual synchrony. On failure of one or more group members, the group membership protocol creates a new group configuration excluding failed members. On recovery of one or more previously failed group members, the group membership protocol creates a new group configuration that includes these recovered members. Finally, the group membership protocol creates a new group configuration when a network partition is restored. This new configuration consists of members of all group configurations that belonged to different network partitions. You must come up with a merge algorithm in this case that merges all queue instances of each queue respecting extended virtual synchrony. Note that you will have to come up with an appropriate configuration naming scheme (similar to version numbering schemes) to be able to merge different partitions.

What to submit

For each phase, please submit a single zip file that contains the following:

- All source code files
- A README file that includes a description of how to compile and run your program. In addition, include the current status of your program – what works, what doesn't, sources of potential errors, etc.

Grading

You will need to demo your program to the TAs during grading interviews and answer questions about your program as well as issues involved in building a group communication system. Each student will have a separate grading interview.