CONCORDIA UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6231, Winter 2023 Instructor: R. Jayakumar

PROJECT

Issued: Mar. 6, 2023 Due: Apr. 9, 2023

Software Failure Tolerant and/or Highly Available Distributed Movie Ticket Booking System

In this project, you are going to enhance either the CORBA implementation (developed in Assignment 2) or the webservice implementation (developed in Assignment 3) of the Distributed Movie Ticket Booking System (DMTBS) to be software failure tolerant and/or highly available under process crash failure using process replication. There are two projects (each team has to select and implement one of the two), suitable for teams of 3 or 4 students, as described in the following.

Project 1 (for teams of 3)

Software Failure Tolerant *or* Highly Available Distributed Movie Ticket Booking System

For this project, extend your Distributed Movie Ticket Booking System implementation from Assignment 2 or Assignment 3 to tolerate either a single software (non-malicious Byzantine) failure or be highly available under a single process crash failure using active replication. Your implementation should be capable of providing either of the required features (software failure tolerance or highly availability) by selecting the feature when the server system is initialized. That is, your server implementation must have the necessary code to tolerate a software failure and be highly available in case of a crash failure and use the appropriate code depending on the failure mode selected when the server is initialized. Your actively replicated DMTBS server system should have at least three replicas each running a different implementation (in different hosts). Each replica has a Replica Manager (RM) which detects and recovers from a failure. You need to implement a front end (FE) which receives a client request and forwards it to a failure-free sequencer. The sequencer assigns a unique sequence number and reliably multicasts the request to all the replicas. In order to handle a software failure, the three replicas execute client requests in total order and return the results back to the FE which in turn returns a single correct result back to the client as soon as two identical (correct) results are received from the replicas. If any one of the replicas produces incorrect result, the FE informs all the RMs about that replica. If the same replica produces incorrect results for three consecutive client requests, then the RMs replace that replica with another correct one. If the FE does not receive the result from a replica within a reasonable amount of time (say, twice the time taken for the slowest result so far), it suspects that replica may have crashed and informs all the RMs of the potential crash. The RMs then check the replica that did not produce the result and replace it with another working replica if they agree among themselves that the replica has crashed. Since the entire server system (replicas, sequencer, FE, and RM) usually runs on a local area network, the server replicas, sequencer, FE, and RM communicate among themselves using the unreliable UDP protocol.

However, this communication should be made reliable in order to avoid message loss. Specifically do the following.

- (*Team*) Assuming that server failure could be due to either a single software bug (non-malicious Byzantine failure) or a single process crash (selected at server initialization), design your actively replicated server system.
- (*Each Student*) Modify the server implementation from the assignment so that it can work as a server replica. A server replica receives the client requests with sequence numbers and FE information from the sequencer, executes the client requests in total order according to the sequence number and sends the result back to the FE.
- (*Student 1*) Design and implement the front end (FE) which receives a client request, forwards the request to the sequencer, receives the results from the replicas and sends a single correct result back to the client as soon as possible. The FE also informs all the RMs of a possibly failed replica that produced incorrect result.
- (*Student 2*) Design and implement the replica manager (RM) which creates and initializes the actively replicated server system. The RM also implements the failure detection and recovery for the required type of failure.
- (*Student 3*) Design and implement a failure-free sequencer which receives a client request from a FE, assigns a unique sequence number to the request and reliably multicast the request with the sequence number and FE information to all the three server replicas.
- (*Team*) Integrate all the modules properly, deploy your application on a local area network, and test the correct operation of your application using properly designed test runs. You may simulate a software failure by returning a random result and a process crash by killing that process while the application is running.

Project 2 (for teams of 4)

Software Failure Tolerant *and* Highly Available Distributed Movie Ticket Booking System

This project is almost the same as Project 1 with the added requirement that your system should be able to work correctly when both a software failure and a process crash occur simultaneously. Thus, the type of failure is not selected when the server system is initialized; rather, the server detects and recovers from both types of failures all the time. In order to do this, your replicated server should have four replicas working as described in Project 1. Specifically do the following.

- (*Team*) Assuming that a single crash failure and a single software failure (Byzantine failure) can occur simultaneously, design your active replication scheme using process group replication and reliable group communication.
- (*Each Student*) Modify the server implementation from the assignment so that it can work as a server replica. A server replica receives the client requests with sequence numbers and FE information from the sequencer, executes the client requests in total order according to the sequence number and sends the result back to the FE.
- (*Student 1*) Design and implement the front end (FE) which receives a client request, forwards the request to the sequencer, receives the results from the replicas and sends a

single correct result back to the client as soon as possible. The FE also informs all the RMs of a possibly failed replica that produced incorrect result.

- (*Student 2*) Design and implement the replica manager (RM) which creates and initializes the actively replicated server subsystem. The RM also implements the failure detection and recovery for both types of failures.
- (*Student 3*) Design and implement a failure-free sequencer which receives a client request from the FE, assigns a unique sequence number to the request and reliably multicast the request with the sequence number and FE information to all the four server replicas.
- (*Student 4*) Design proper test cases for all possible failure situations and implement a client program to run these test cases.
- (*Team*) Integrate all the modules properly, deploy your application on a local area network, and test the correct operation of your application. You may simulate a process crash by killing that process while the application is running and a software failure by returning a random result.

Timeline and Marking Scheme

Before implementation, you should submit your design documentation by Monday, March 20, 2023, and get the TA's approval. This is a DESIGN project; if you implement a bad design, even though it may work, you'll still lose up to 50% of the marks.

[30%] Design Documentation (by group). ***Due by Monday, March 20, 2023.***

- [20%] Describe and explain your design and architecture clearly, including theories (protocol, algorithm) you apply, how to implement, and also describe dataflow (how your modules interact/cooperate with each other to achieve the function).
- [10%] Design proper and sufficient test scenarios, which should include testing data and results.
- Please indicate the student ID of each student, the module that student is responsible for and how that module is designed and integrated.
- Keep the documentation ready to display and discuss during your demo.

[70%] Demo. ***Monday, April 10, 2023 and Tuesday, April 11, 2023.***

- Please select your preferred demo time (20 minutes per group) in advance. There is no instant registration during the demo week, so if you cannot register before demo week, you will lose 40% of the mark.
- Make sure your application can work on a network (you may use workstations from a lab) during demo; otherwise you'll lose 70% of the mark. There will be no second chance.
- [50%] For group demo:
  - Introduce your application architecture.
  - Demo your designed testing scenarios to illustrate the correctness of your design. If your testing scenarios do not cover all possible issues, you'll lose part of mark up to 40%.

- [20%] For individual demo:
  - Introduce the module you are responsible for and answer questions on your module.

QUESTIONS

If you are having difficulties understanding any aspect of this assignment, feel free to contact your teaching assistants (Lab FI: Anurag Shekhar anurag.shekhar@mail.concordia.ca, Lab FJ: Kiana Nezami nezami.kiana@gmail.com, Lab FK: Yash Patel ycpatel1999@gmail.com). It is strongly recommended that you attend the lab sessions, as various aspects of the assignment will be covered there.