CS 6378: Programming Project II

Instructor: Ravi Prakash

Assigned on: October 8, 2019
Due date: October 29, 2019

This is an individual project and you are required to demonstrate its operation to the instructor and/or the TA to get credit for the project.
In this project, you are required to implement a quorum system as described below.

1 Requirements

1. Source code must be in the C/C++/Java programming language.
2. The program must run on UTD lab machines (`dc01, dc02, ..., dc45`).
3. You will need to know thread and/or socket programming and its APIs for the language you choose. It can be assumed that each process (server/client) is running on a single machine (`dcXY`). Please get familiar with basic UNIX commands to run your program on *dcXY*.

2 Description

1. There is a fileserver, S_0, hosting only one file, file.txt. Initially, the file is empty.
2. There are seven servers, numbered from one to seven, arranged as a binary tree. Consider S_1 as the root node of the tree. Each server S_i in the tree is the parent node for servers S_{2i} and S_{2i+1}.
3. There are five client nodes, numbered from one to five. Each client can enter the critical section by executing the following sequence of operations. Each client should generate twenty SATISFIED requests.
   (a) The client should wait for a random amount of time in the range between 2 and 5 seconds before trying to enter the critical section.
   (b) Then, it sends a REQUEST message to a quorum of servers and waits for GRANT messages from them. The recursive logic of selecting a quorum of servers in a binary tree of servers is as follows:
      i. Pick the root node of the tree + Select a quorum of left subtree, or
      ii. Pick the root of the tree + Select a quorum of right subtree, or
      iii. Select a quorum of left subtree + Select a quorum of right subtree.
      iv. If the tree is a singleton node (a node with no child), then select that node as the tree's quorum.

      Using this logic, multiple quorums are possible in this system of seven servers. Each time a client makes a REQUEST, it should re-run the quorum selection algorithm which randomly generates one of the quorums to send the REQUEST to. Although the number of possible quorums is limited, you are not allowed to hardcode the quorums. It is intended to select different quorums over the course of this project.

      Once all the server nodes in the chosen quorum have GRANTed the REQUEST, the initiator client enters the critical section.
   (c) The critical section execution consists of the following procedures:

i. The client sends its request, which contains its client ID and physical time, to the fileserver S_0.

ii. Upon receiving the request, S_0 opens file.txt and writes "request from" followed by the received client ID and the received time in a new line at the end of the file. Then, it closes the file.

iii. Once S_0 is done writing to the file, it sends a success message to the requesting client.

iv. The client waits for a random amount of time in the range between 1 and 3 seconds, and then exits the critical section by sending RELEASE message to all servers in the chosen quorum.

(d) Each client upon exiting the critical section for twenty times will send a completion notification to the fileserver S_0.

4. The servers operation is as follows:

   (a) Initially, the state of all servers is UNLOCKED.

   (b) If a client's REQUEST is received by an UNLOCKED server, the server goes into LOCKED state and sends a GRANT message back to the client.

   (c) If a client's REQUEST is received by a LOCKED server, the server adds the REQUEST to its queue ordered by the physical timestamp assigned to the REQUEST by the corresponding client.

   (d) When a client's RELEASE message is received by a server, if the server is LOCKED by that client, the server first checks its queue. If it is not empty, the server dequeues the REQUEST with the lowest physical time from the queue, sends a GRANT message to the corresponding client of the REQUEST and stays in the LOCKED state. Otherwise (queue is empty), the server becomes UNLOCKED.

5. S_0 brings the entire distributed computation to an end once it has received completion notification from all the clients.

3 Data Collection

The following outputs should be reported, either on the screen or written to a report file:

1. The total number of messages sent by each node, either server node or client node, from the beginning until it sends the completion notification.

2. The total number of messages received by each node, either server node or client node, from the beginning until it sends the completion notification.

3. For each attempt to enter the critical section by each node, report the following:

   (a) The total number of messages exchanged.

   (b) The total elapsed time between making a request and being able to enter the critical section (latency).

4. Rerun your experiment with other values of: (a) time between a client exiting its critical section and issuing its next request, and (b) time spent in the critical section. Report what impact do these changes have on performance, specifically latency. Also, report if you come across any deadlock situation.

4 Point Distribution

Implementation (50%): Source code of your well structured and well documented program.
Correctness (50%): Output that your program produces and the performance data and its analysis.

5 Submission Information

The submission should be through eLearning in the form of an archive consisting of:

1. File(s) containing the source code. Your source code must have the following, otherwise you will lose points:
   (a) Proper comments indicating what is being done.
   (b) Error checking for all function and system calls.
2. The README file, which describes how to run your program.