CS/CE/TE 6378: Advanced Operating Systems
Section 002
Project 3

Instructor: Neeraj Mittal

Assigned on: Thursday, November 3, 2016
Due date: Thursday, December 1, 2016

This is a group project. *Code sharing among groups is strictly prohibited and will result in disciplinary action being taken.* Each group is expected to demonstrate the operation of this project to the instructor or the TA.

You can do this project in C, C++ or Java. Since the project involves socket programming, you can only use machines dcXX.utdallas.edu, where XX $\in$ {01, 02, .., 45}, for running the program. Although you may develop the project on any platform, the demonstration has to be on dcXX machines; otherwise you will be assessed a penalty of 20%.

1 Project Description

Implement Koo and Toueg's checkpointing and recovery protocol. Develop a simple application to demonstrate the working of the protocol.

You should design your program to allow *any node to initiate an instance of checkpointing or recovery*. However, you can assume that at most one instance of checkpointing/recovery protocol will be in progress at any time. For example, if node 2 has initiated an instance of checkpointing protocol, then no other node can initiate an instance of checkpointing or recovery protocol until the instance initiated by node 2 has terminated.

You will be provided with a sequence of checkpointing/recovery operations you have to simulate in a configuration file. The sequence will be given by a list of tuples; the first entry in a tuple will denote a node identifier and the second entry will denote an operation type (checkpointing or recovery). As an example, if the list of tuples is (2,c), (1,r), (3,c), (2,c) then your program should execute an instance of checkpointing protocol initiated by node 2, followed by an instance of recovery protocol initiated by node 1, and so on.

Avoiding Concurrent Instances: To ensure that only one instance of checkpointing/recovery protocol is in progress at any time, use the following approach. Once the current instance of checkpointing/recovery protocol terminates, its initiator informs the initiator of the next instance of the checkpointing/recovery protocol (of the termination of the current instance) using simple flooding. The latter then waits for instanceDelay before initiating the appropriate protocol. In the previous example, node 2 should inform node 1 which in turn should inform node 3 and so on.

Contents of a Checkpoint: Each instance of checkpointing protocol will have a sequence number associated with it. A node, when taking a checkpoint, stores: (a) the sequence number of

the checkpointing protocol, (b) the current value of its vector clock (you will need to implement vector clock protocol for this), and (c) any other information you may deem to be necessary for application's recovery.

Modeling Application: We will model an application using a single parameter sendDelay, which denotes the delay between two consecutive send events of a node. To execute a send event, a node selects neighbor uniformly at random and sends a message to that neighbor.

Testing: Show that your checkpointing and recovery protocols work correctly. Specifically, for the checkpointing protocol, you have to show that the set of last permanent checkpoints form a consistent global state. For the recovery protocol, you have to show that the protocol rolls back the system to a consistent global state.

2 Submission Information

All the submission will be through eLearning. Submit all the source files necessary to compile the program and run it. Also, submit a README file that contains instructions to compile and run your program.

3 Configuration Format

Your program should run using a configuration file in the following format:

The configuration file will be a plain-text formatted file no more than 100KB in size. Only lines which begin with an unsigned integer are considered to be valid. Lines which are not valid should be ignored. The configuration file will contain $2n + m + 1$ valid lines.

The first valid line of the configuration file contains five tokens. The first token denotes the number of nodes in the system. The second token denotes the number of checkpointing/recovery operations you have to perform. The third token denotes the mean value of minInstanceDelay (in milliseconds). The fourth token denotes the mean value of minSendDelay (in milliseconds). The fifth token denotes the number of messages that a node should generate. Assume that both instanceDelay and sendDelay are *exponentially distributed* random variables.

After the first valid line, the next n lines consist of three tokens each. The first token is the node ID. The second token is the host-name of the machine on which the node runs. The third token is the port on which the node listens for incoming connections.

The next n lines consist of the up to n tokens each. The first token is the node ID. The next up to $n - 1$ tokens denote the IDs of the neighbors of the node referred to by the first token. You can assume that the neighborhood information in the configuration file will be symmetric.

Finally, the next m lines consist of a tuple each, where each tuple denotes a checkpointing/recovery operation that you have to perform.

Your parser should be written so as to be robust concerning leading and trailing white space or extra lines at the beginning or end of file, as well as interleaved with valid lines. The # character will denote a comment. On any valid line, any characters after a # character should be ignored.

You are responsible for ensuring that your program runs correctly when given a valid configuration file. Make no additional assumptions concerning the configuration format. If you have any questions about the configuration format, please ask the TA.

Listing 1: Example configuration file

```
5 10 100 10 2000

0 dc02.utdallas.edu 1234
1 dc03.utdallas.edu 1233
2 dc04.utdallas.edu 1233
3 dc05.utdallas.edu 1232
4 dc06.utdallas.edu 1233

0 1 2
1 0 3
2 0 3
3 1 2 4
4 3

(c,1)
(c,2)
(c,0)
(r,2)
(c,4)
(c,1)
(c,0)
(r,3)
(c,3)
(c,2)
```