CS 6378: Project III

Instructor: Ravi Prakash

Assigned on: April 9, 2013
Due date: April 30, 2013

This is an individual project and you are expected to demonstrate its operation either to the instructor or the TA.

You are required to implement the following *tree-based voting protocol* for replica consistency. Let there be seven servers, $S_0, S_1, S_2, \ldots S_6$, that maintain replicas of data objects. The seven servers are logically arranged as a balanced binary tree as shown in Section 6. There are four data objects, $D_0, D_1, \ldots, D_3$ of type integer. Each server maintains copies of the four data objects, along with their version number. Initially, all the replicas are consistent, with D_j initialized to $3 - j$ with version number equal to 1, where $0 \leq j \leq 3$. There are five clients, $C_0, C_1, \ldots, C_4$. All communication channels are FIFO.

The protocol for data access is as follows:

1. When a client, C_i wishes to access a data object D_j,[1] it sends a request of the form REQUEST(C_i, D_j, v) to all seven servers in the logical tree and starts an AWAITING_GRANT timer. The purpose of this access operation is to add v to the current value of D_j, where v is an integer. The value of the AWAITING_GRANT timer is equal to 20 time units (given in the config file in Section 6).

2. A server can grant only one request at a time for a data object. If a server has not granted any request for that data object when it receives C_i's request, then the server sends a GRANT message to C_i and enters a blocked state corresponding to that object. Subsequent requests received by the server for the same object, while it is blocked by C_i, are placed by the server in a queue corresponding to that object.

3. If client C_i's request has been granted by the tree of servers before the expiry of its AWAITING_GRANT timer then the client does the following: (i) first C_i waits for a period of time referred to as the HOLD_TIME, then (ii) client C_i sends a COMMIT message to all servers. The HOLD_TIME is equal to 1 time unit. Granting of the request by the tree is recursively defined as:

   (a) The request has been granted by the root of the tree, and either the left or the right subtree, OR

   (b) The request has been granted by the left subtree and the right subtree.

   If a subtree has only one server, then the granting of request by that subtree is equivalent to obtaining a grant from that server.

4. On receiving the COMMIT message from C_i, all the servers perform the operation indicated in the corresponding REQUEST message, increment the version number of the targeted data object by one, send an ACK to C_i, and remove C_i's request from their queue. If the server was blocked by C_i's request then the server gets unblocked on performing the data update operation. Now, it can grant the request at the head of the queue of pending requests for the newly updated object.

5. If a requesting client's AWAITING_GRANT timer expires before it receives permission from the tree then the client withdraws its request by sending a corresponding WITHDRAW message to all servers, and increments the number of unsuccessful accesses by one. The variable to store the number of unsuccessful accesses is maintained locally at each client, and is initialized to zero.

6. On receiving a WITHDRAW message from C_i, servers perform the same operation as on receiving the COMMIT message, except for performing the access operation.

[1] For simplicity we consider all accesses to be write operations.

If you believe that this protocol may result in writes to a data object being performed at different servers in different order, add safeguards to prevent such a possibility.

1 Simulating Failure and Recovery

Client C_0 has the special ability to selectively deactivate and reactivate servers. This capability will be used to simulate failure and subsequent recovery of servers. C_0 will send the deactivate and reactivate messages to server(s) only when it is not trying to access a data object. The following messages will be used for this purpose:

- DEACTIVATE(S_i): When this message, sent by C_0, is received by server S_i the server empties all its queues, and silently discards all other messages until it receives a REACTIVATE message.
- REACTIVATE(S_i): When this message, sent by C_0, is received by server S_i the server should poll other servers to get the latest values of all the data objects from the live servers, update its copies of the data objects to reflect this information, and resumes participating in the voting and data update operations. *You need to design and implement a protocol for the recovering server to obtain the latest values from the live servers.*

2 Operation with No Failure

1. A client can have at most one pending access request at a time. The time between the completion (successful or unsuccessful) of the previous access and the next access attempted by a client is uniformly distributed in the range [5,10] time units. Use the same distribution for the initial access. When a client wishes to perform an update, it arbitrarily selects one of the four data objects for the update and initiates the protocol as described above.
2. In your experiments all communication should be performed using IP stream sockets.
3. Execute your experiment until a total of M updates have been attempted.
4. Repeat the experiment with the HOLD_TIME set to 0.1, 0.5, 1.5, 2.0, and 5.0 time units.

3 Operation with Failure and Recovery

1. After client C_0 has completed $\frac{M}{5}$ (successful + unsuccessful) update operations it sends DEACTIVATE messages to servers listed in the config file (see Section 6).
2. After client C_0 has completed $\frac{2M}{5}$ (successful + unsuccessful) update operations it sends REACTIVATE messages to the same set of servers.

4 Data Collection

For each experiment report the following:

1. The number of successful and unsuccessful accesses by each client.
2. The total number of messages exchanged.
3. For the successful accesses, the minimum, maximum, average, and standard deviation of the time between issuing an access request and receiving permission from the server tree.
4. For every data object, do all replicas of the object go through exactly the same sequence of updates?
5. What is the impact, if any, of the value of HOLD_TIME on the performance of the protocol?

5 Submission Information

The submission should be through *elearning* in the form of an archive (.zip, .tar or .gz) consisting of:

1. File(s) containing only the source code.
2. The README file, which describes how to run your program.

NOTE: Do not submit unnecessary files.

6 A sample config file

```
# A sample configuration file
# Any text following # should be ignored
# Number of Servers connected in binary tree topology
NS=7
#     0
#    / \
#   1   2
#  / \ / \
# 3  4 5  6
#
# Number of Clients
NC=5
# Total number of requests sent for objects
M=500
# Time unit in millisecond
TIME_UNIT=100
# Nodes IP and port addresses
SOCKINFO:
0 net20.utdallas.edu 3333
1 net21.utdallas.edu 3336
2 net22.utdallas.edu 4444
3 net23.utdallas.edu 3334
4 net24.utdallas.edu 5334
5 net25.utdallas.edu 5334
6 net26.utdallas.edu 5334
7 net27.utdallas.edu 5647
8 net28.utdallas.edu 5647
9 net29.utdallas.edu 6456
10 net30.utdallas.edu 8967
11 net31.utdallas.edu 9067
# List of nodes that will fail after M/5 requests
FAILINGNODES:
1 3 5
```