CS 6378: Project III

Instructor: Ravi Prakash

Assigned on: November 17, 2011
Due date: December 6, 2011

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, $S_1,\ S_2, \ldots S_7$, that maintain replicas of data objects. The seven servers are logically arranged as a balanced binary tree with S_1 being the root, and servers S_{2i} and S_{2i+1} being the left and right children, respectively, of server S_i. There are four data objects, $D_0,\ D_1,\ D_2,\ and\ D_3$ of type integer. Each server maintains copies of the four data objects. Initially, all the replicas of an object are consistent, with D_j initialized to j, 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.

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 access operation indicated in the corresponding REQUEST message, send an acknowledgement 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 serves in different order, add safeguards to prevent such a possibility. You need to instrument your code such that if these safeguards get triggered, a corresponding message is displayed on the screen.

1 Operation

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 five 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 500 updates have been attempted.

4. For the experiment report the following:

   (a) For every data object, do all replicas of the object go through exactly the same sequence of updates?

   (b) The number of successful and unsuccessful accesses by each client.

   (c) The total number of messages exchanged.

   (d) For the successful accesses, the minimum, maximum, and average time between issuing an access request and receiving permission from the server tree.

5. Repeat the experiment with the HOLD_TIME set to 0.1, 0.5, 1.5, 2.0, and 5.0 time units.

6. What is the impact, if any, of the value of HOLD_TIME on the performance of the protocol?