Project 2 Design Document and Performance Results

Abhijeet Gaikwad
Prashant Chaudhary

Communication:
We use RMI to communicate between the servers and client-server. Detailed interface: PROJ_HOME/common/src/main/java/edu/umn/bulletinboard/common/rmi/BulletinBoardService.java

Client:
Client starts a shell, these commands can be used at command line:

-- connect server_ip server_port
Used to join to RMI server <server_ip> on port <server_port>. This connection is provided so that Read Your Write consistency can be tested.

-- post article
Posts an article to the server. New id assigned is displayed on the console.

-- reply id article
A new article is posted as a reply to article with id : `id`. New id is returned and displayed on console. Reply is allowed without reading because clients usually tend to know what they have posted. This may not be always true but we have considered it as an assumption.

-- read
Prints all the articles on the Bulletin Board server. Output is properly indented to show article-reply mappings. An augmented DFS traversal is used to indent the articles, we level of each node in the tree in the stack. After popping we indent level number of tabs.

-- choose id
Choose an article with id as `id` from the articles displayed as result of read call. Choose will fail if read was never called before. Once a read is called all subsequent choose calls will be successful irrespective of sequence of operations until disconnect is called. This is an assumption.

-- disconnect
Disconnects the client from the server connected using connect call as explained above.

Few Notes:
A server details are given as argument to client, client will not pick up nearest server which would be a bit ambitious.

Server:
This section aims at explaining the Server implementation and decisions that were taken while implementation that directly impact the performance of the operations. The number of servers that can be handled currently is 97 excluding coordinator. Quorum based system will be inconsistent if servers are more than this value.

Coordinator:
This is a type of role that a Server can enact. Currently client can contact coordinator if it has the RMI Ip and Port. This is because coordinator cannot distinguish between Server and a client. One of the easiest ways to distinguish is to pass a magic number as an argument which will identify a server. For the purpose of the assignment it is assumed that client will never call coordinator. Coordinator is assigned id 99 (each server is assigned a value internally).

MemStore:
An article object contains: id, article text and list of replies. We have an in memory article store. This article store is a Map of `id -> Article` which provides constant access to a particular article. This map is used for doing DFS traversal.

Sequential Consistency:

Write: A request is sent to the server which contacts coordinator. Primary backup strategy is used. Data is written to all the servers by the coordinator. Please note that if simultaneous connections are made clients may get stale data. To ensure consistent data please make requests after the client writing the articles has finished executing. It finishes when control is returned to user on Command line client.

Read: Read is easy, whichever server the client connects to will be most updated.

Quorum Consistency:

Write: A request is forwarded to coordinator which starts the quorum write. Randomly Nw servers are picked up by coordinator and then the data is updated at these servers.

Read: A request is forwarded to coordinator which starts the quorum read. Randomly Nr servers are picked up by the coordinator and contacted for latest Article id that they have. Server than identifies the max article id and gets the data from that server. This data is returned to the client as a string: in ready to display form.

Sync: Sync is called after every 30 seconds in the background. Data is collected from Server with the latest update and broadcast to all the servers. After this operation all servers will have up to date data.

Read Your Write Consistency:

Write: Write operation writes the data to the server. It also maintains a cache. All the data that was written by the client is retrieved from the cache when read is called. This makes sure that the client always gets the articles that it has written previously. On exit all of the cache is cleared.

The strategy currently uses primary-backup so writes are expensive but reads on same client are highly optimized.

Read: A client will have a cache, so the data is returned locally from the client cache. This improves performance as no call is made to the server. However, as we do not clear the cache intermittently, it is bound to use up a lot of memory. We assume that sufficient memory is available on client.

Sync: As we have used primary backup in the backend to synchronize the servers we do not explicitly synch.

Overall Notes, assumptions:

-- Read call does not show trimmed output. It shows whole text of the article. This is a disadvantage as more data is transferred over the wire that degrades performance.

-- All server to server communication sleeps for random time ranging between 200 - 300 ms. This is to emulate wide-area networks.

-- Currently all the articles are displayed on console in one go. No paged display is provided.

Performance Runs:

We performed experiments with different consistency models and recorded corresponding read and write times.

Results for read times

Consistency Type	Average Read	Read time 1	Read time 2	Read time 3
QC NW=4 NR=3	1465.285556	1361.651167	1534.294321	1499.91118
QC NW=5 NR=1	1007.58800933333	976.282632	1029.218436	1017.26296
QC NW=3 NR=5	1931.744244	1958.624431	1944.178312	1892.429989
Sequential	244.847346666667	281.068115	237.165784	216.308141
ReadYourWrite	1.87126533333333	4.398447	0.604461	0.610888

Results for write times

Consistency Type	Average Write	write time 1	write time 2	write time 3
QC NW=4 NR=3	1504.78453066667	1517.234229	1506.307161	1490.812202
QC NW=5 NR=1	1824.217552	1806.412858	1813.351352	1852.888446
QC NW=3 NR=5	1242.74233633333	1180.758882	1268.561427	1278.9067
Sequential	1844.607746	1795.434666	1795.329996	1943.058576
ReadYourWrite	1775.733225	1693.079707	1795.003647	1839.116321

Graph showing the comparison of average read and write times