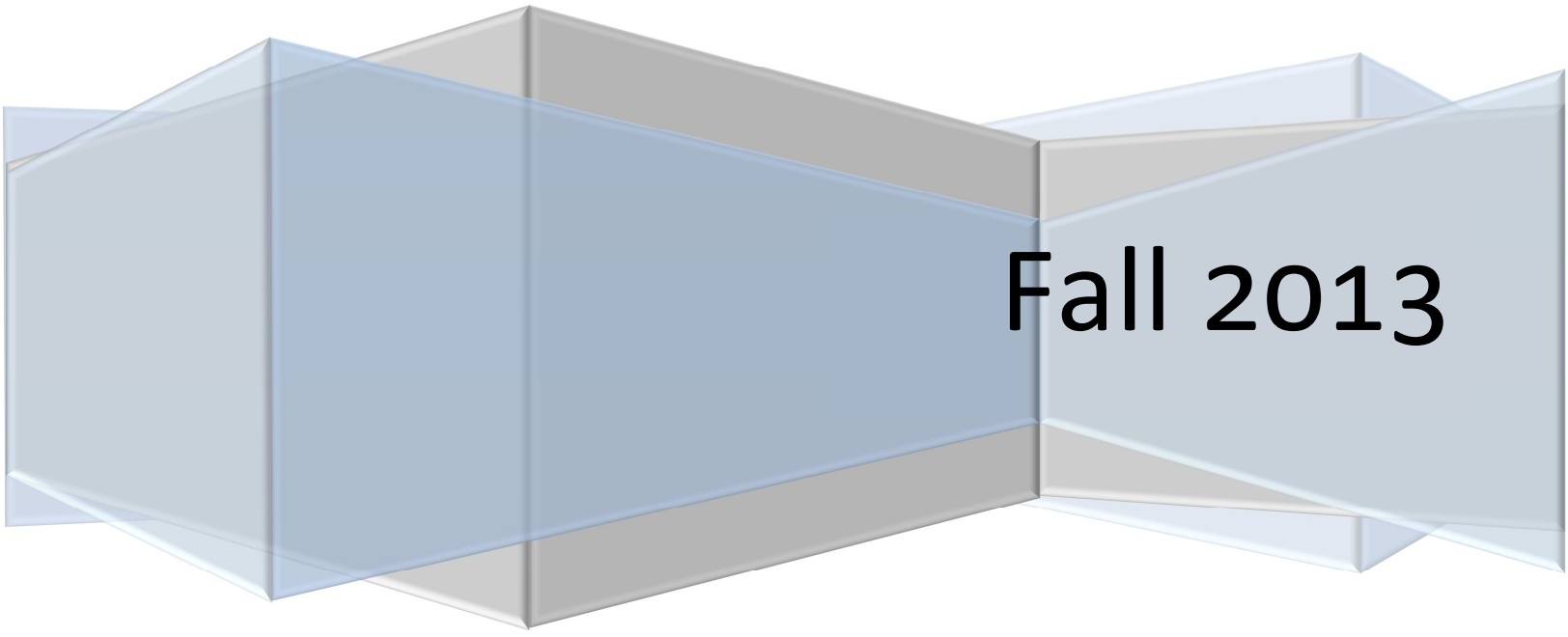
The University of Texas at Dallas

AOS Project3

Voting Protocol

By Sanket Chandorkar and Ashish Gupta



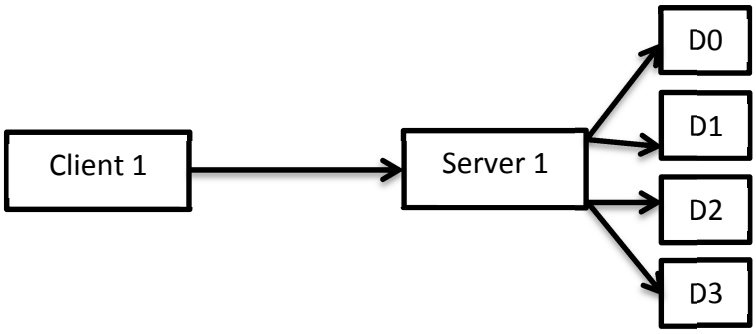
1. Objective:

To implement the following tree-based voting protocol for replica consistency.

2. Design Overview

Some Design Decision/Assumptions:

1. Since the communication channel is reliable and FIFO, we will use **TCP/IP protocol** for communication.
2. Message queue for client/server will be implemented using a **producer-consumer** like design.
3. **Java** will be the language used for implementation.
4. **Messages** will be implemented as java objects.
5. **Factory pattern** can used to implement **message generation.**
6. Each client and server service will be running a request listener for service that will service incoming requests. **Singleton pattern** will be used for client and server service.
7. Each Client and server runs on a separate thread and each data object access at the server is maintained by a dedicated thread for that data object. This will ensure that request for different data objects at a given server are serviced concurrently.
8. Thus the server will act like a dispatcher; which dispatches appropriate request to respective dataObjects message queue.



Figuure1: Client – Server- DataObject Communication.

(NOTE: Each box is a thread)

3. Class Diagram

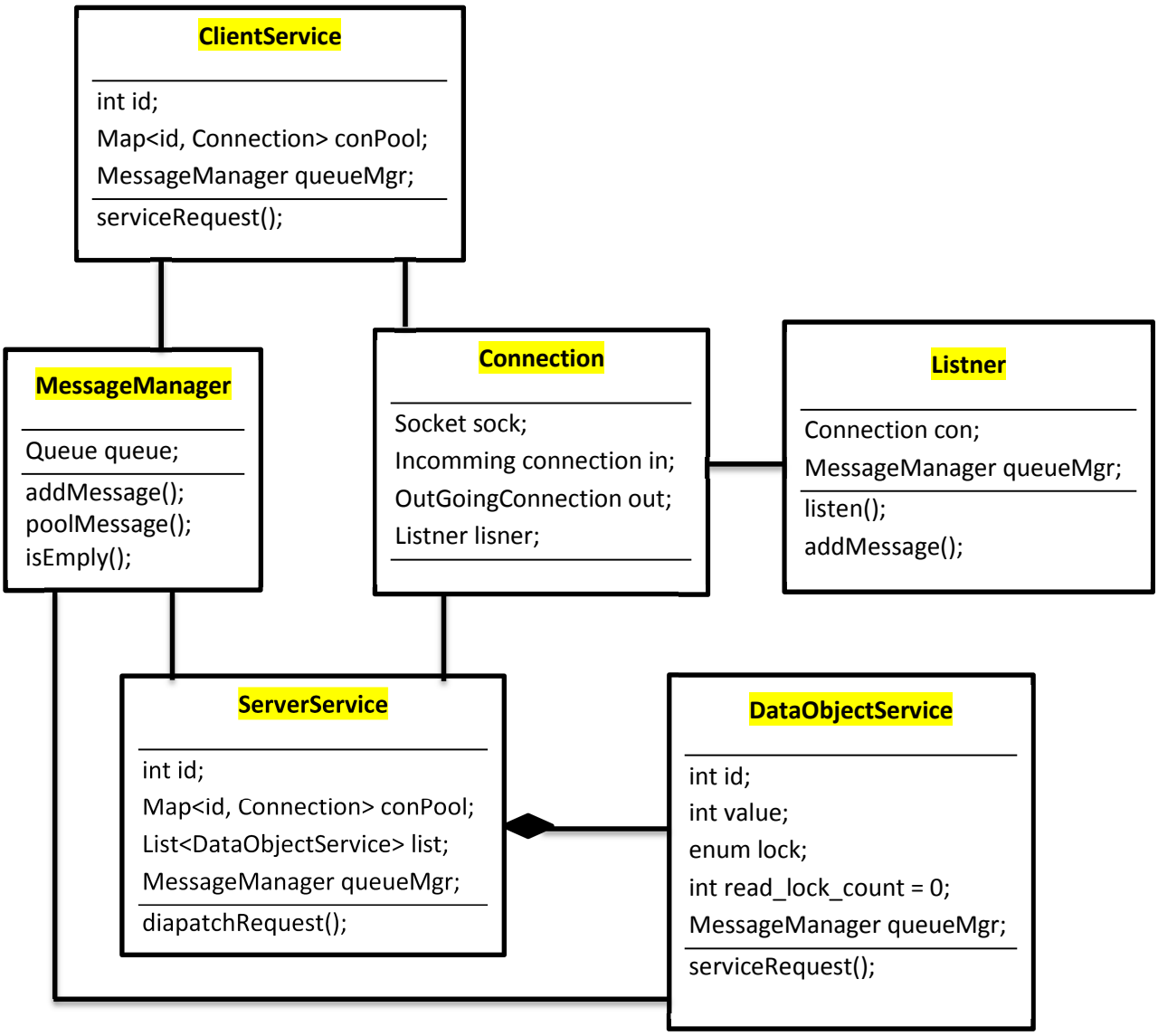


Figure 2: Class Diagram

4. Project Planning (Contribution / Responsibility)

The project was implemented in following phases:

Phase	Description	Work Hours Required	Contributor
Phase 0	**Understand** the Problem statement.	4 Work Hours	**Both**
Phase 1	**Design**: Identifying major modules and how they work together.	8 Work Hours	**Both**
Phase 2	**Implementation**		**-**
	1. Common modules	8 Work Hours	**Both**
	2. Client Module	10 Work Hours	**Ashish**
	3. Server Module	10 Work Hours	**Sanket**
	4. Integration	4 Work Hours	**Both**
Phase 3	**Testing**		**-**
	1. Client Test	2 Work Hours	**Ashish**
	2. Server Test	2 Work Hours	**Sanket**
	3. Integration Test	1 Work Hours	**Ashish**
Phase 4	**Report**	3 Work Hours	**Sanket**
Phase 5	**Result Collection**	1 Work Hours	**Both**

Total Work Hours Required to implement project: 52 Work Hours

5. Implementation

The Project implementation involved implementing the following classes.

1. **Client**:
    a. **ClientDriver:** Acts as the starting point for the client program.
    b. **ClientService:** Issues requests and services the incoming requests/replies to the client.
    c. **Timer:** A Thread that performs timeout operation after 20 Units of time.
2. **Server:**
    a. **ServerDriver:** Acts as the starting point for the client program.
    b. **ServerService:** Issues requests and services the incoming requests/replies to the client.
    c. **DataObjectService:** A Thread that performs timeout operation after 20 Units of time.
3. **Message:** Classes required for **message** passing, like READ_REQUEST, WRITE_REQUEST message.
4. **Supporting**:
    a. **Message Manager:** Manages/Allows synchronized access to message queue.
    b. **CommonAPIs:** API used by both client and the server.
    c. **Connection:** Data structure that maintains attributes related to connection with a server/client.
    d. **ConnectionAcceptor:** Accepts Incoming connections.
    e. Listener Thread: Listens for incoming requests on a connection and adds the message to the MessageQueue (Maintained by MessageManager)

6. Testing

The Project was tested in parts/phases.

1. **Module Level testing:**
    Each of the client and the server module were tested separately by stubbing the other module.
2. **Integration testing:**
    Integration testing was performed by integrating both the client and the server modules.

7. Results:

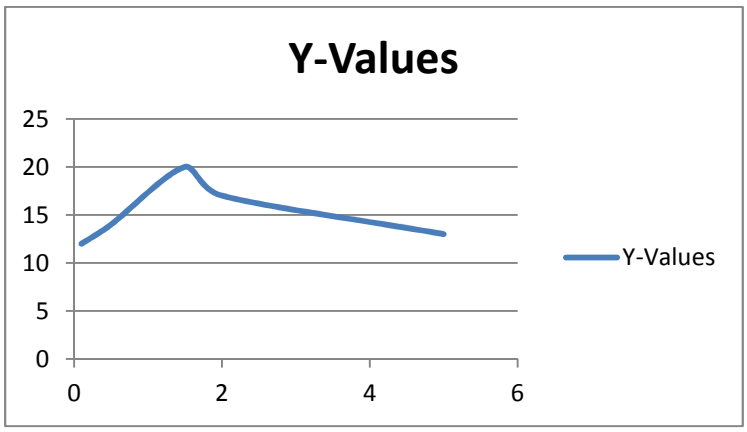
(Sample result) (HT = HOLD TIME)

Sr no	Query	HT = 0.1	HT = 0.5	HT = 1.5	HT = 2.0	HT = 5.0
1	For every data object, do all replicas of the object go through exactly the same sequence of updates?	yes	yes	yes	yes	yes
2.1	The number of successful READ accessed.(Success/Total)	219 / 229	216 / 221	208 / 220	209 / 219	218 / 222
2.2	The number of successful WRITE accessed.(Success/Total)	12 / 21	14 / 29	20 / 30	17 / 31	13 / 28
3	The total number of messages exchanged.(Total)	1152	1395	1499	1333	1275
4	For the successful READ accesses, the minimum, maximum, and average time between issuing a READ request and receiving permission from the requested server.	MIN= 0.1 AVG= 3.5 MAX= 19.5	MIN= 0.1 AVG= 5.2 MAX= 18.5	MIN= 0.1 AVG= 4.3 MAX= 19.7	MIN= 0.1 AVG= 3.7 MAX= 19.7	MIN= 0.1 AVG= 2.3 MAX= 17.4
5	For the successful WRITE accesses, the minimum, maximum, and average time between issuing a WRITE request and receiving permission from the server tree.	MIN= 1.8 AVG= 4.1 MAX= 7.6	MIN= 0.2 AVG= 3.6 MAX= 5.8	MIN= 0.3 AVG= 3.3 MAX= 18.5	MIN= 0.7 AVG= 4.2 MAX= 12.0	MIN= 0.2 AVG= 1.3 MAX= 4.6

8. Analysis:

1. Effect of Hold Time(SUCCESSFUL WRITE):

X Axis = Hold Time
Y Axis = Successful writes

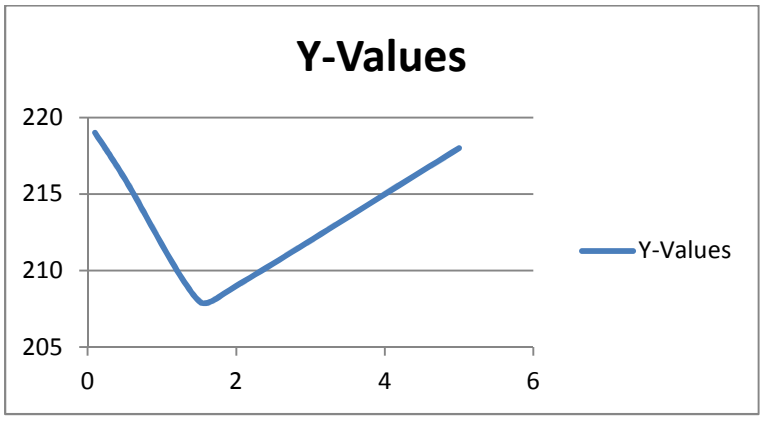


Effect of Hold Time on UNSUCCESSFUL WRITE

UNSUCCESSFUL WRITE will increase with increase in HOLD_TIME(After peak as shown above).

2. Effect of Hold Time(SUCCESSFUL READ):

X Axis = Hold Time
Y Axis = Successful reads



3. Special Scenarios (SAFEGUARDS):

1. Handling WITHDRAW message at SERVER

In the scenario when time out occurs at client, client sends the WITHDRAW message with respective REQUEST_ID. At server two cases can happen:

1. If the server had granted this request it will withdraw it by updating its LOCKS
2. If the server was not involved in grant to this request (i.e request was still in queue), the REQUEST_ID from the WITHDRAW message received will help the server to identify which request to remove from the queue.

2. REQUEST ID

In cases shown below the grant message will have the REQUEST-ID as sent by the client and that will be used by the client to skip the grant message received after the timeout occurred for the current request.