CMPE 275 – Fall 2017
Prof. John Gash

Project 1 Submission Report

RollBits

Distributed Chat System

Akansha Mehta (011820679)
Dhrumil Shah (011428859)
Nishant Rathi (011812788)
Rashmi Sharma (011818729)

RollBits is a distributed chat application. It is designed to send and receive messages from the servers in internal and external clusters in an expedited manner.

Functionalities:

As discussed in class we have implemented following functionalities:

1. Register a user (No validation for pre-existing users)
2. Send a message from a user to another user
3. Register a group (With validation across clusters).
4. Add a user to group (With validation across clusters)
5. Send message from a user to a Group
6. Check messages for a user both personal and group messages

Design Approach:

Created Peer-to-Peer application that consists of following components:

(1) *Network Discovery via UDP*:

We have Java client and server running on each node. The client on each sender node builds the message which consists of its node details (Node Name, Node IP, Node Port, Cluster Name, Mode, Sender Type(External/Internal)) and sends the datagram packet to broadcast address (255.255.255.255) and port (8888) through netty channel. The servers running on each node reads the channel, stores the node's details, builds and sends a response message to server of the sender node.

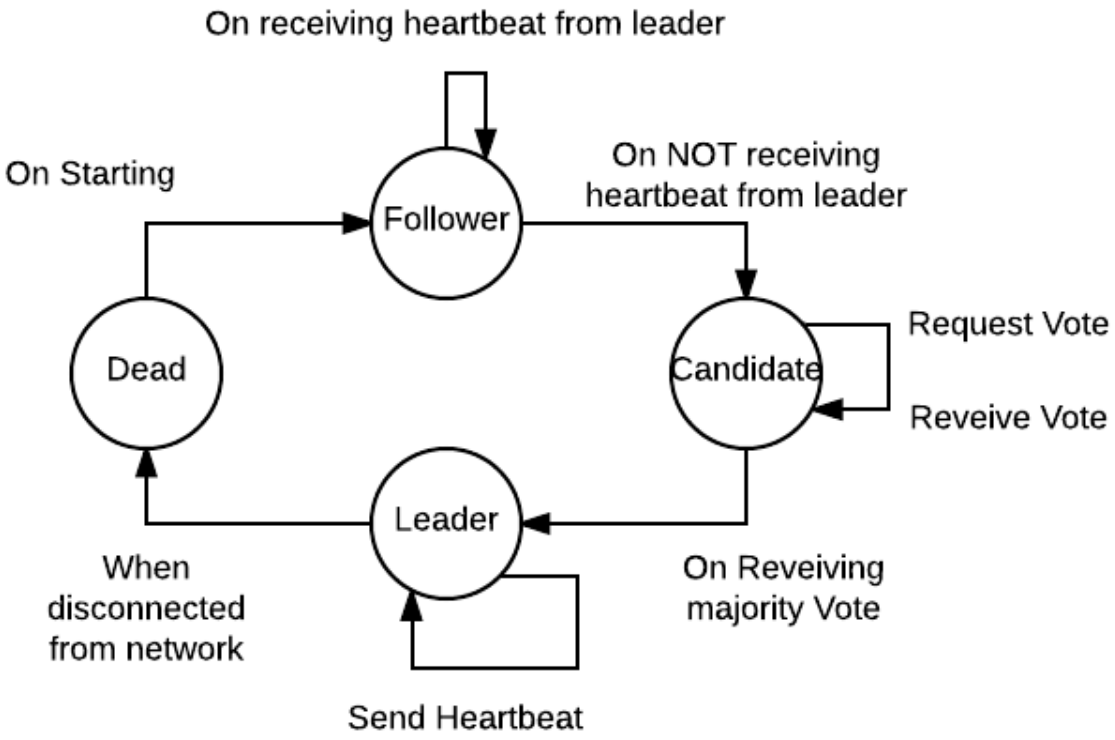
1. UDP server is up and continuously listen to following type of packets:
    A. New Node up broadcast
    B. Node removal request
2. Whenever a node goes up, a packet is broadcasted with the details including node name, IP, port, cluster group tag etc.
3. Periodically broadcast your presence in the network
4. Maintain a cluster directory

(2) *Raft Algorithm*:

We have implemented Raft consensus algorithm. It has been implemented in java using state pattern. In our system, we need leader for following purpose:

1. Monitor the health of all the nodes in the system using less number of heartbeat type packets.
2. If a node goes down, intimate all the node in cluster.
3. If a node goes down, intimate all other clusters by UDP broadcast to remove the node that went down.
4. Whenever a node goes down, it triggers the failover mechanism and re-adjust the sharding and replication mechanism of the system across all nodes (as ours is a

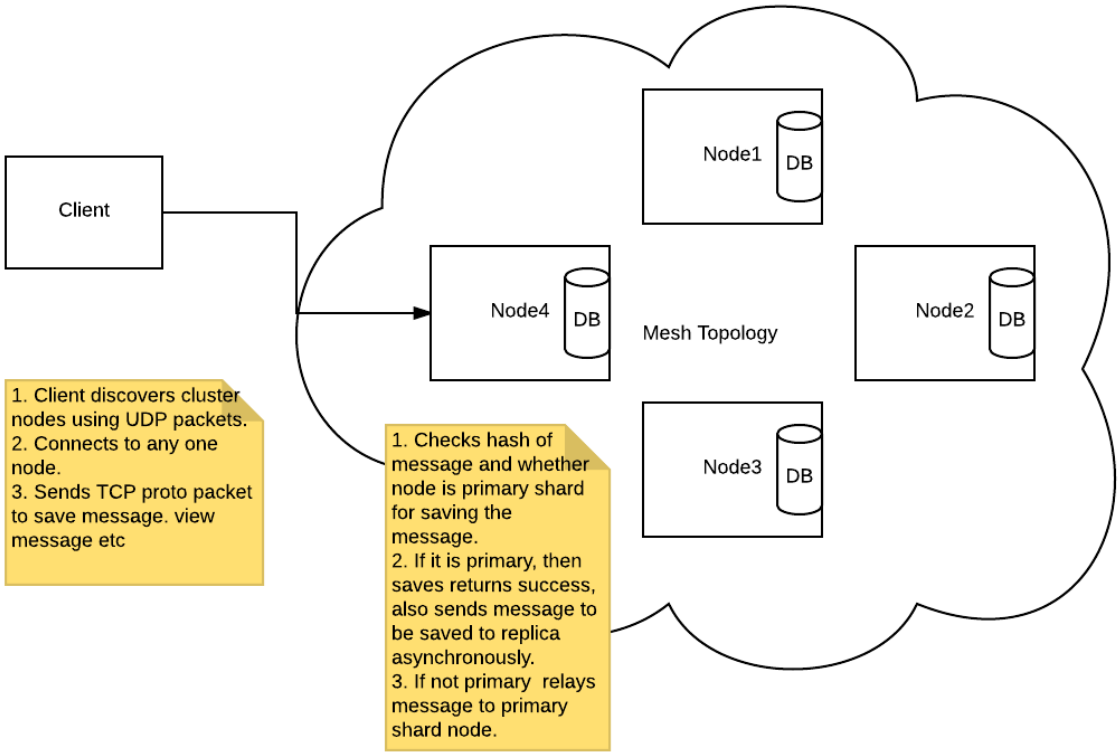
p2p kind of system and external client can contact any of our node, so each node should know about new sharding and replication nodes).



(3) Sharding and Replication.

Since this is messaging application, we forecasted high volume of data in future, hence we decided on using Consistent Hashing based sharding, so that we can easily scale storage capacity and utilize current capacity.
We have given paramount importance to **eventual consistency,** as we consider this as non-mission critical application, wherein delay in receiving messages can be tolerated.
Due to above reason, adding messages to primary shard is important and replicating to replica is asynchronous in nature, rather than following strict quorum based approach.

A. Sharding: As soon as the network goes up, each node in the cluster forms a logical ring. As the node knows about all other nodes present in the node, logical ring is formed individually in every node.
B. Replication: We have parameterized replication factor and have it currently set it to 2. One is primary node and the other one is replica node.
C. Failover: As we have leader elected by raft, it intimates all the nodes whenever there is a failure in the cluster. The logical ring is formed again.



(4) Inter-cluster communication

For group-related functionalities: As discussed and finalized keeping the scope limited, data related to a group will always be stored in same cluster. If external client requests some other cluster, the request will be forwarded to the owner cluster and hence served. Hence, we have implemented two inter cluster services:

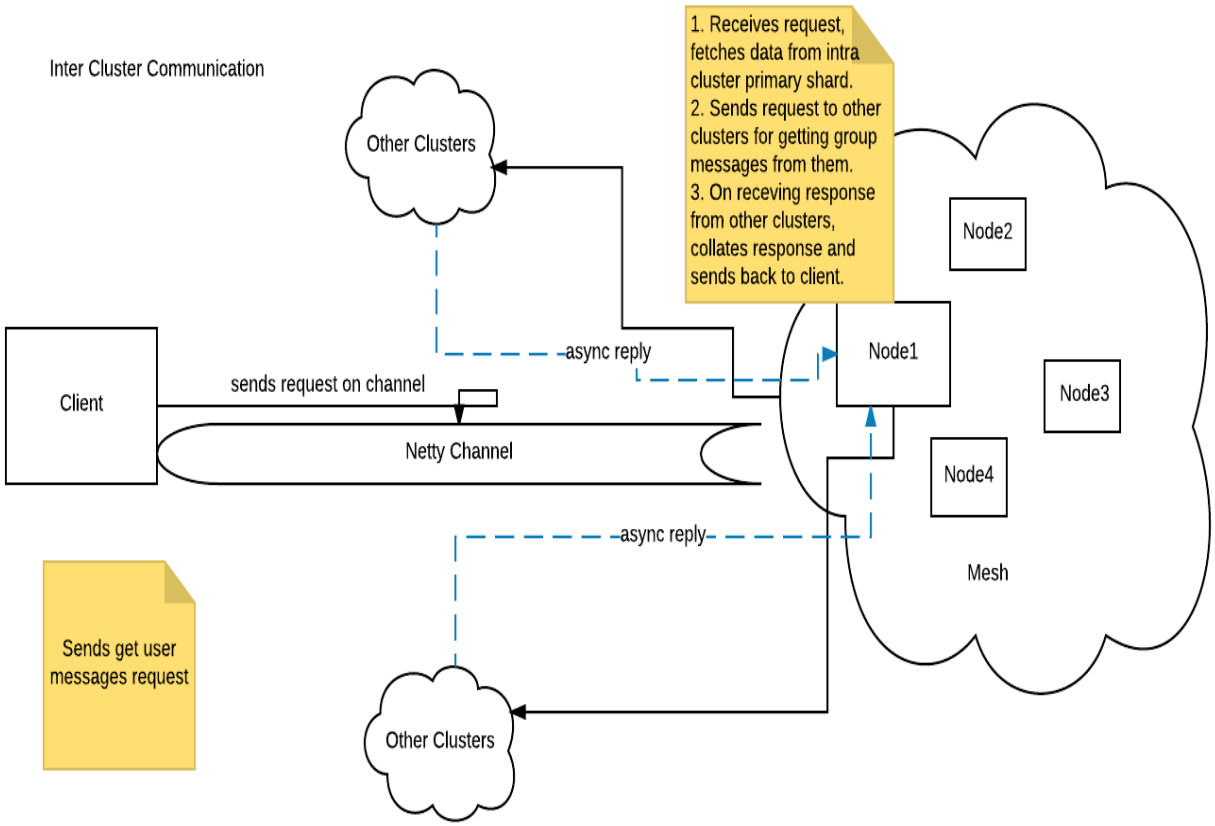
1. To add user to a group
2. To send a message to a group

If group is owned by the cluster receiving the request, it is stored in the same cluster itself. If not, the request is forwarded to other clusters. The cluster owning the group does the tasks and return success message, which is collected and the user is intimidated.

For Check Message functionality: This request will get personal messages as well as messages on his group(s). All the cluster will be contacted and messages will be collated and returned to external client.

1. Check the messages for a user

If a user requests to fetch all his messages, the cluster receiving will fetch personal and group messages stored in its cluster. Also, all other cluster is requested and asked for messages (both personal and group). The messages are then collaborated and send back to user in response.

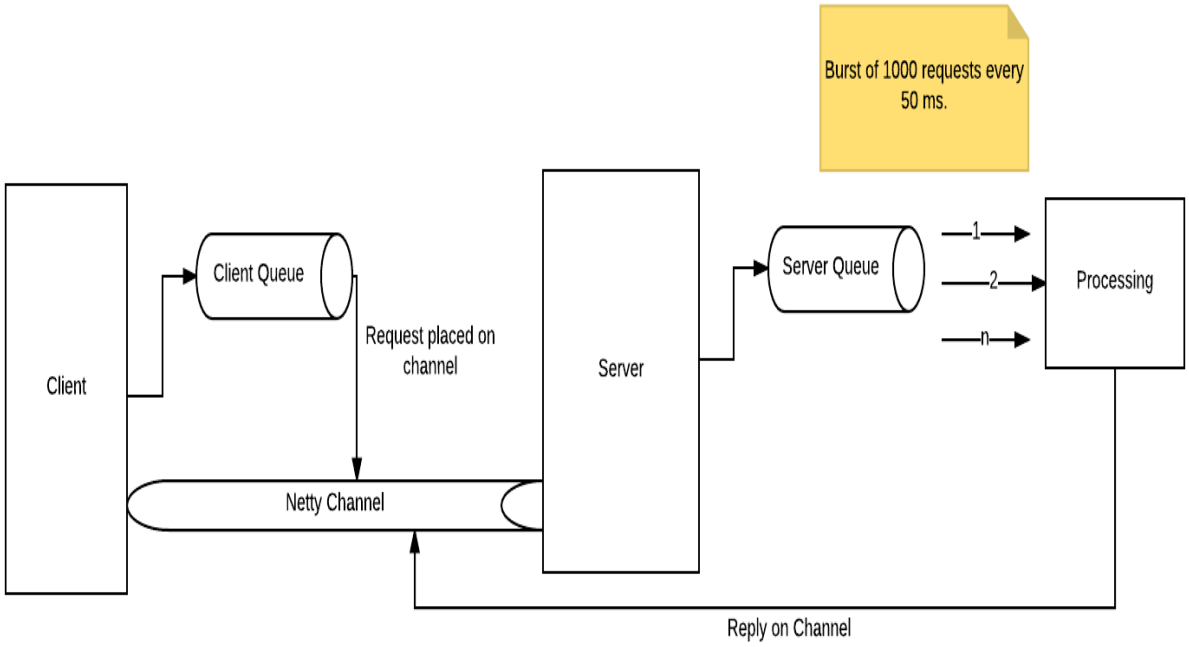


Scaling Issues/ Testing Changes:

On performing load test, we found below issues:

Too Many Open Connections on netty channel:

1. Fixed above issue by increasing ulimit on linux systems, while on MacOS Sierra, increasing ulimit didn't help much.
2. Used Queue at server end, thereby releasing netty server thread early and passing on the request to fixed thread pool (50). After, 50ms threads will serve requests accumulated in queue. We have used burst size=1000 i.e. after 50ms threads will serve 1000 messages and wait for next burst time.



Database Connection Errors:

1. Fixed Database connection issues by using, hibernate session factory, c3po connection pooling and batch commits.

Test Results

Test No	No of Messages	Time in Milliseconds (Message send—> DB Save—> Response)	Throttling If any
1	25000	20202	50ms after 100 records
2	25000	17442	50ms after 1000
3	25000	12747	50ms after 2000
4	25000	11936	50ms after 5000
5	25000	11303	Without throttling
6	100K	39067	stopped after 58000 messages in 39067 ms without throttling
7	100K	58636	50ms after 5000
8	500K	219606	50ms after 10000

External Team Coordination Challenge

It was a big challenge coordinating with external teams, but we could finally overcome that challenge by agreeing to:

1. work on a common proto file. Each team contributed to the proto file and integrated it with their work.

2. follow a unique range of static IPs, common subnet mask and port for UDP discovery.

Fault Tolerance

The advantage of our application is that it does not contain any single point of failure. Our application contains four servers and leader is elected through raft consensus algorithm. The leader keeps track of the status of all the other servers (whether dead or alive). When the leader itself dies, the leader election process starts once again and the new leader is elected. Thus, our application is not dependent on any single server or single node. Moreover, in our consistent hashing algorithm, we are maintaining the replicas of every node and thus, in case of failure of leader or any node, the data is maintained in the replicas thus, preventing data loss of all the nodes.

Contributions:

Akansha
1. UDP discovery
2. Group Dao and Group Service Layer
3. Intra Cluster Communication Implementation for Group
4. Performance & Functionality testing

Dhrumil
1. DAO & Service Layer
2. Intra Cluster Communication Implementation for Adding User to Group
3. YAML Configuration Load changes
4. Performance & Functionality testing
5. External client Implementation

Nishant:
1. Raft Implementation
2. Inter cluster Services Communication
3. Performance & Functionality testing
4. Server Side Queue Implementation
5. External client Implementation

Rashmi
1. Sharding and Replication using Consistent Hashing
2. Intra Cluster Communication Implementation for Message, User, Group Message
3. Performance & Functionality testing
4. Inter cluster Services Communication
5. Netty Connection and channel handling

README

To deploy the project

1. To start the application, you need to run MessageApp.java

2. This class does the following:
   2.1 Starts UDP Server.
   2.2 UDP Broadcasts its details.
   2.3 Starts Raft Context.
   2.4 Starts Server Request Queue.
   2.5 Starts the netty TCP Server.

3. The following configuration files has been used:
   3.1 routing.conf: Contains details regarding tcp port and routing resources .
   3.2 config.yml: Contains constants used project wide including node details like name, IP, group tag etc.
   3.3 hibernate.cfg.xml: Contains mysql and hibernate related configurations.

Code Walkthrough

1. UDP: com.sjsu.rollbits.discovery
2. Raft: com.sjsu.rollbits.raft
3. Sharding & Replication: com.sjsu.rollbits.sharding.hashing
4. DAO: com.sjsu.rollbits.dao.interfaces
5. InterCluster Services: com.sjsu.rollbits.intercluster.sync
6. External Client: com.sjsu.rollbits.client