SAN JOSÉ STATE
UNIVERSITY

Team Project

Project-1

CMPE 275– Enterprise Application

Submitted To: Prof. John Gash
Date of Submission: 04/12/2013

Submitted By:
Gaurav Kesarwani (009278815)
Sarvagya Jain (009269377)
Ayush Mittal (009311068)
Swapnil Pancholi (009303684)

Contents

Introduction :

A distributed system is a software system in which components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal. Three significant characteristics of distributed systems are: concurrency of components, lack of a global clock, and independent failure of components.

All distributed systems have to deal with maintaining consistency in one way or another. To simplify, the strategies may be divided into two groups. The ones that attempt to avoid inconsistencies from occurring and the ones that define how to reconcile them.

The aim of the project is to provide a flexible network to support the adhoc nature of collaborative learning i.e. developing collaborative MOOCs. In order to achieve this, we are using Netty framework, which provides Java library and asynchronous APIs for developing non-blocking client-server network application. Our client is developed using Python, server using java and Netty 4.0, and for the database we are using RIAK, which is a NOSQL database.

System Architecture

- ## Client Server Communication

The following figure shows 3-tiered high-level system architecture of the project. The client provides the functionalities such as getting list of courses available in MOOC, get the course description based on course IDS, sign in/sign up Voting etc. The server serves the client request with the help of database. It takes the client request, process among the group of servers, retrieves/checks the data from database and provides the response to client. The communication between client and server/ server-server is taking place using Google protobuf. the client is sent to server by means of Google protocol buffer. The protocol buffer supports all the three languages mentioned above.

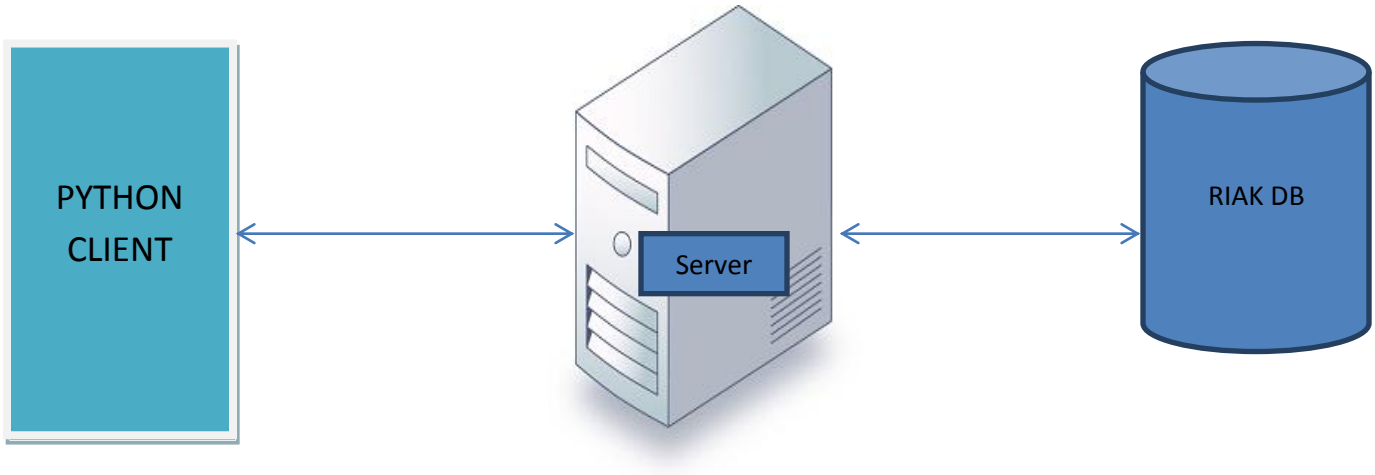


Figure 1: 3-Tiered High Level System Architecture

At the server side we have four core Netty servers which can interact with each other to transfer the packet to the right destination. Each server has an inbound and outbound queue. The request is received on inbound queue and after processing reply is put on the outbound queue. The servers interact with Riak database to store and fetch data. Riak database manages sharding and replication automatically in a distributed environment.

- ## Overlay Network

Overlay network refers to a network that is built on top of one or more existing networks. We have created an overlay network where all the nodes are connected through logical links in the fashion as shown below:

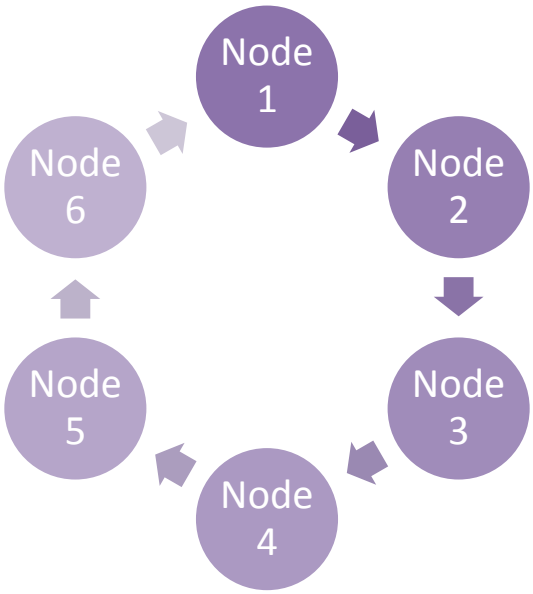


Figure 3: Network Topology

The network topology used for connecting all the servers in MOOC is Ring topology. Each server has one neighbor where it can forward its request to. The routing takes place on the basis of the request sent by the client. Request processing is handled as:

```
RequestHandling() {
        if (server is destination node) {
Complete the request operation ()
        else
```

Forward request to neighboring node

Technologies Used

- ## Netty:

As Wikipedia states "**Netty** is a non-blocking I/O (NIO) client-server framework for the development of Java network applications such as protocol servers and clients. The asynchronous event-driven network application framework and tools are used to simplify network programming".

- **Why use Netty?**

Some of the Netty features that describe why Netty is useful for building communication system are:

i. An asynchronous Non-blocking IO API.
ii. provides low level data transfer and mechanism
iii. No response is generated, while invocation is instantaneous and response is generated in another thread
iv. Works through listeners and events.

- **What is NIO and how it differs from OIO?**

Old IO was a stream of bytes and the new IO is a block of bytes, old one was a synchronous while this one is Asynchronous. NIO is good for the situations where you have to keep a lot of connections open and do a little work amongst them, like in chat applications. In Old IO you keep thread pools to manage lot of thread IO work.

- **Netty terminology (Rough overview)**

i. ***Channel:*** a abstraction for communication or transfer of data (byte array)
ii. ***ChannelBuffer:*** Byte array which can provide random or direct data storage mechanism.
iii. ***ChannelPipeline:*** Collection of handlers to process your events in the way you want.
iv. ***ChannelFactory:*** Resource allocators, like thread pools.
v. ***ChannelHandler:*** Provides method to handle events on channel.
vi. ***ChannelEvents:*** Events like connected, exception raised, and message received etc.

Google Protobuf :

Protocol Buffers are a way of encoding structured data in an efficient yet extensible format. Protocol Buffers are a method of serializing structured data. If we want to communicate with others applications, irrespective of the language of the recipient application or to send data over wire, or to store data, Google Protobuf message is an efficient way to do so. After looking at other serialization techniques, Protobuf seemed to be a natural, efficient, easy and extensible format.

- Why not use other serialization techniques?
    i. **Use Java Serialization:** This is the default approach since it's built into the language. It doesn't work very well if you need to share data with applications written in C++ or Python. Moreover, it breaks when class definition changes.

    ii. **Serialize the data to XML:** We can use XML if we want to share data with other applications/projects. However, XML is space intensive, and encoding/decoding it can impose a huge performance penalty on applications. Also, navigating an XML DOM tree is considerably more complicated than navigating simple fields in a class normally would be.

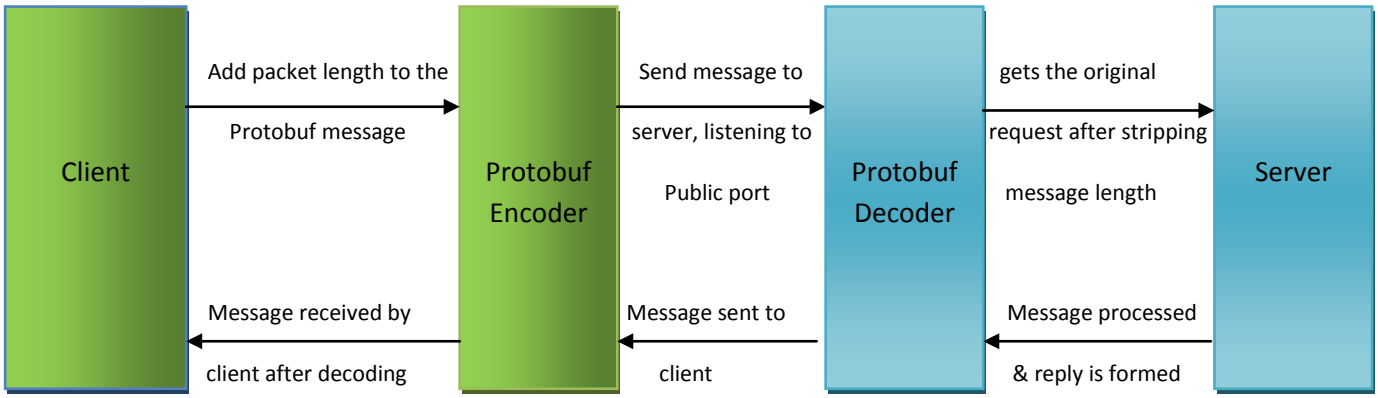


Figure 2: Client-Server communication using Google Protobuf in the project

i. **Ad-hoc encoding/decoding: Writing** an ad-hoc way to encode the data items into a single string. It will require encoding and parsing code, and the parsing might include a small run-time cost. This works best for encoding very simple data but get complex with complex objects and more time has to be invested in its encoding/decoding.

- ## RIAK Database

It is a key-value NOSQL, open source, distributed database.

- **Why only RIAK and not any other database?**
  Out of the few options, that we considered we found that RIAK would be best for our project as it is a distributed system and having a ***Ring*** topology. This provides us with few functionalities that we do not have to worry about. Its main features are:
  i. **Fault-Tolerance:** It distributes the data across nodes using ***consistent hashing*** in a simple key/value scheme in namespaces called buckets. Therefore, we can lose access to one to many nodes due to network partition or hardware failure without losing data.
  ii. **Availability**: It maintains replication and data sharding by itself. The data can be inserted and retrieved intelligently so it is available for read and write operations on any node in distributed system, irrespective to the node where data was inserted.
  iii. **Scalability**: Riak evenly distributes data across nodes with consistent hashing and and yields a near-linear performance increase as you add capacity.
  iv. **Operational Simplicity**: We can add/delete new machines to our Riak cluster without incurring a larger operational burden. The same operational tasks apply to small clusters as well as large clusters.

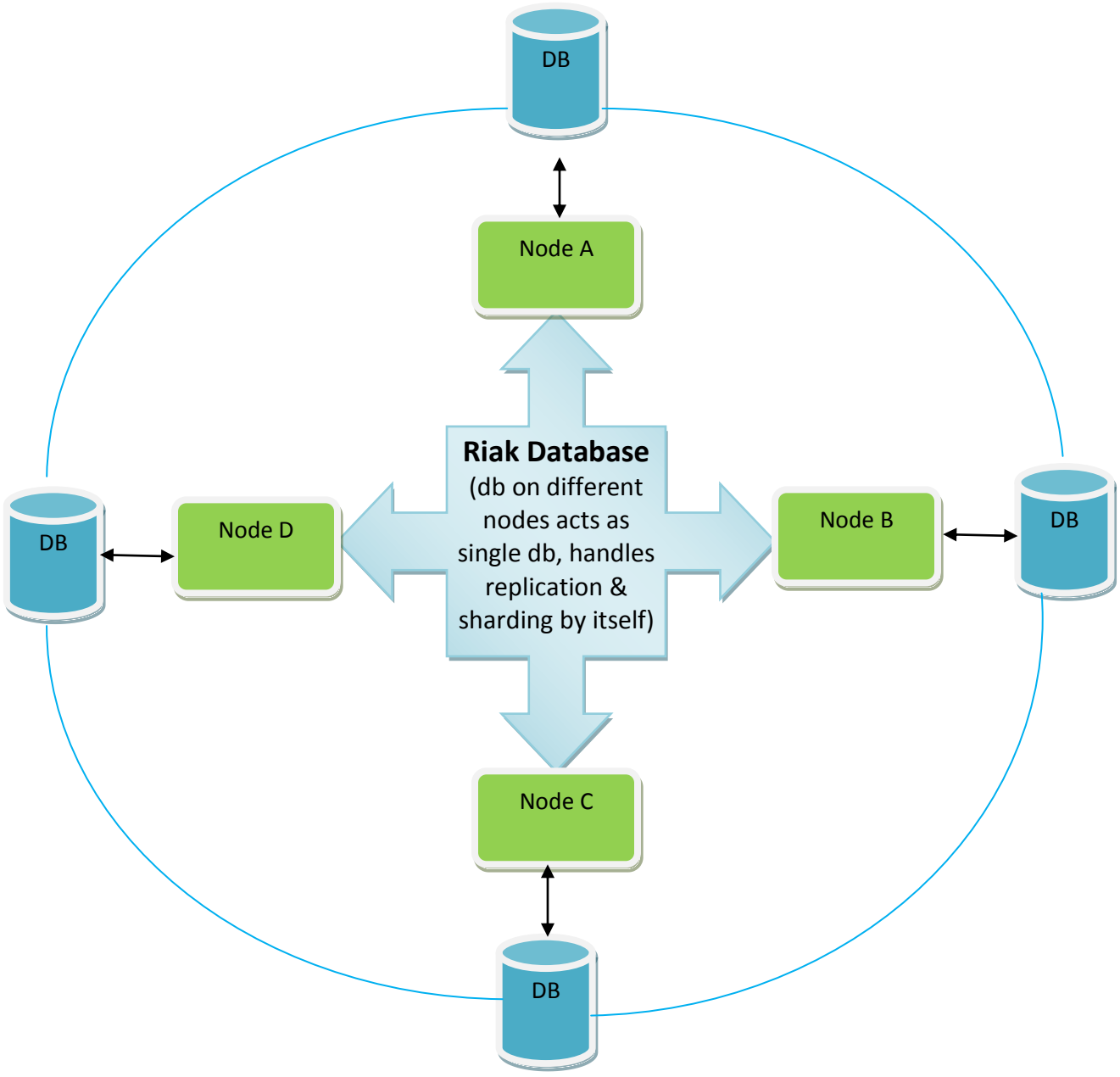


Figure 3: Riak Database implemented in project, database present on each node but acts as a single one, handles replication & sharding through concurrent hashing in the cluster

Design features

Public Communication

I. **Client**- The client part of the project is implemented in Python. We have used a heterogeneous environment equivalent to the real world scenario, Python client interacts with Java Platform. We have used the python buffer API to read & write message. We have added a message in .proto file for each data structure we want to serialize, then specified a name and a type for each field in the message that we would send it to Netty Server. We send the message through server public port and serialize it using length-prefix framing. It is efficient in both space as well as time. We read the message from the socket. In this manner, the communication between client & server takes place.

II. Request mechanism:

Whenever a request is sent from client, only leader can listen to that request and respond.

A request consists of request header and request body. Request header is built by header parameters (Header tag, timestamp, Router Id, Originator, ToNode). Request body comprises of the payload that is generated by combining namespace, signin, signup, getcourse, requestlist, initvoting etc.

III. Response mechanism:

Response, as in case of request, also consists of header and body. For generating response, first step is to set request header

IV. Forward request:

Forward request is a design feature that forwards the request to the attached neighbor node if it is not the destination node.

The request when reaches to the server, it checks for a a particular attribute in the header of the request, 'toNode', which means that the request is mean to be processed by that particular node.

i. If the request contains 'toNode' or has some value in this attribute, then the server checks whether its own id matches with toNode's value:
   a. If it matches, then the request is processed by the server
   b. else the message is passed on to the neighboring node.

   The process continues till the request reaches its destination, i.e. the value to 'toNode'.

ii. If toNode does not contain any value in the request, then the leader assigns some node to process the request. The leader server fills in the toNode value. Then the process continues as mentioned in (i).

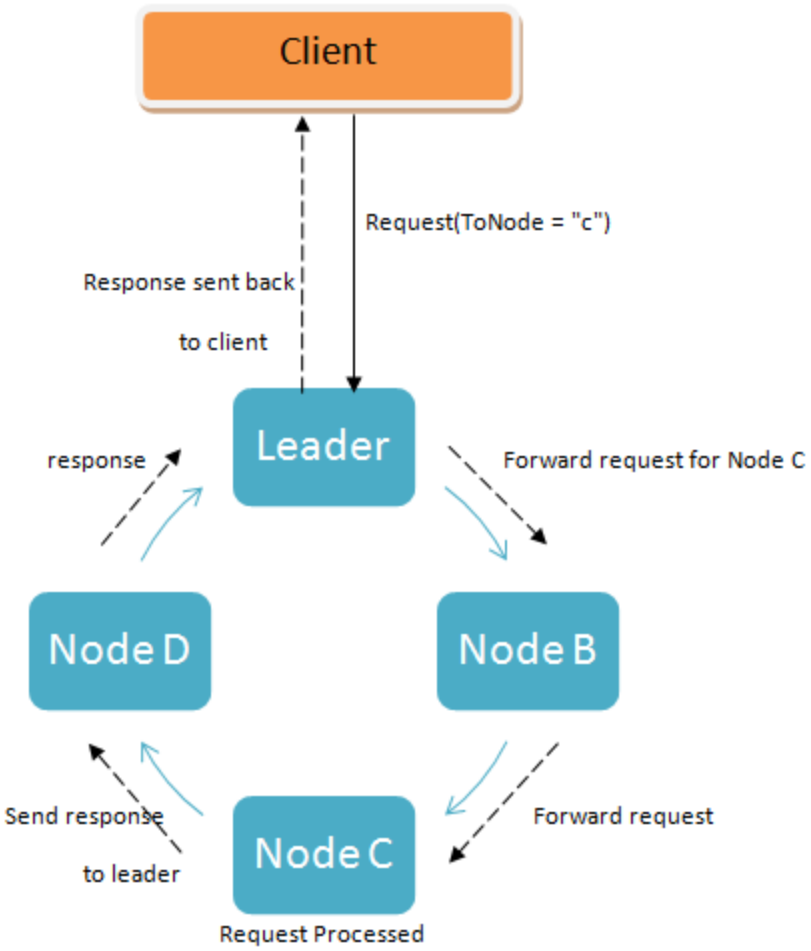


Figure: How to request sent by client is processed at the server

Management Process

1. **Heartbeat Process :**

   The internal management of the servers is being performed through a separate management channel. We have implemented a unidirectional ring topology among the cluster of four nodes. Each server keeps track of its nearest neighbor through a heartbeat message. When node A is connected to node B, node A starts receiving a heartbeats from node B. We have implemented fault tolerance to ensure that if a node(say node C) goes down in the ring topology then the network node(B) reconfigures and connects to its nearest node(node D) so that ring is complete again.

2. **Leader Election :**
    a. **Why Leader Election?**
    In a cluster of servers or say distributed system, reason for selecting a leader is similar to keyword synchronized in Java. As in Java, race condition arrives if two threads try to access same object's member variable. Similarly, in cluster if we consider thread as a node, on arrival of a request, two nodes might process the same request and it might lead to corrupt data, wastage of resources etc. Leader election algorithms are designed to be economical in terms of total bytes transmitted, and time.

    b. **Our implementation :**
    We are following a distributed architecture in which we are selecting a leader among the cluster nodes via **LCR leader election algorithm**. The leader is responsible for assigning the incoming request from client to a node in the cluster.
    We have implemented LCR Leader Election Algorithm in ring topology. Each node starts sending Leader nominate message to its neighbor node once it starts receiving heartbeats from the neighbor node. The node with highest node Id is elected as leader. Each server has an Election Manager who is responsible for processing of the leader election message. The election manager decides whether to discard the message or forward the message to its nearest node. Once the election manager of the sending node receives its own message it declares itself as leader.

3. **Intra-Cluster Voting :**
    This section describes how the voting is implemented within the mooc. In this every node's decision is taken into consideration and the final decision is taken according to the majority of votes.
    When the leader receives a request for voting, it initiates the voting process. Since, it is a ring topology; this is how we are implementing voting within our cluster. When a server receives a voting request, it adds his own vote and increases the no of nodes participating in election by 1 and forwards the message to the neighbor node. The next node follows the same process. When leader receives back the request of voting, with all the votes included, it decides voting results by giving the result in the favor of majority. The decision takes place with everybody's consensus.

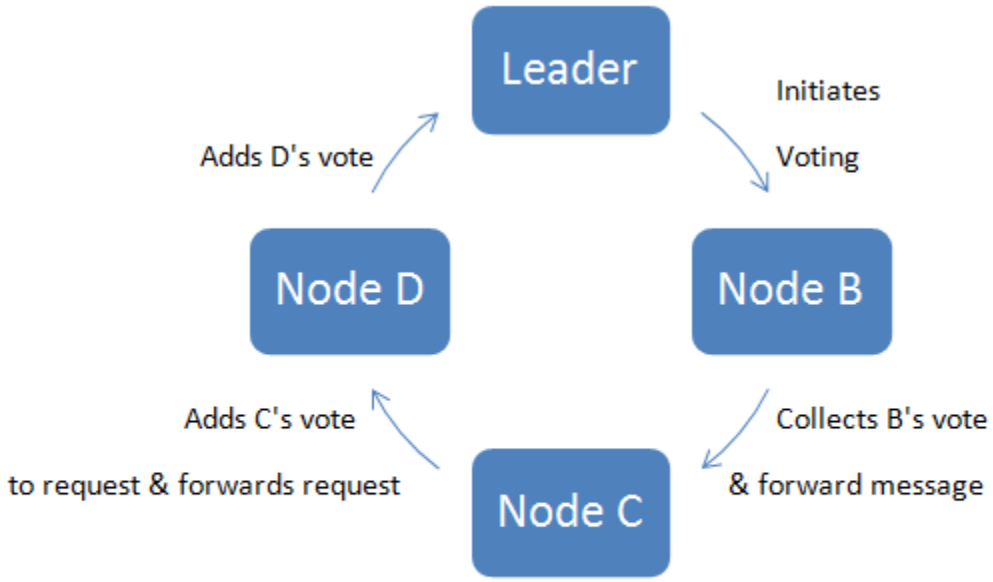


Figure : Decides voting result depending on majority

4. **Inter Cluster Voting :**
   We are also implementing the inter cluster communication in order to decide the host cluster of a particular competition. The host cluster is decided by the voting algorithm implemented among the clusters.
   When a cluster receives a request for hosting a competition among various MOOCs. It gets the leader info from all the MOOCs and sends a voting request to all the MOOCs. Then inter MOOC voting takes place and they reply back with the response.

Limitations

- Each packet of data has to pass through all the other nodes to reach the destination.
- If one node goes down the entire network gets affected

Challenges

- We faced challenges in routing the messages between nodes from source to destination. Another major challenge was to make the network fault tolerance to ensure that network connectivity is established among the existing nodes if any of the nodes

goes down. We have implemented the functionality to re-establish the heartbeat connectivity between existing nodes in case if any of the node becomes faulty at any time. Though due to time limitations we could not implement the functionality to re-initiate the leader election algorithm after the network connectivity is re-established.

Screenshots

Leader Election:

Server2:

```
[inbound-mgmt-1] INFO management - Discard the election message if the id is lesser::one
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-2-2] INFO management - Received HB response from three
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Forward Node Id ::three
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Discard the election message if the id is lesser::one
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-2-2] INFO management - Received HB response from three
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Forward Node Id ::three
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Discard the election message if the id is lesser::one
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-2-2] INFO management - Received HB response from three
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Forward Node Id ::three
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Discard the election message if the id is lesser::one
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
```

Server 3:

```
[inbound-mgmt-1] INFO management - Leader Elected :: three
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Discard the election message if the id is lesser::two
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-2-1] INFO management - Received HB response from zero
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Leader Elected :: three
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Discard the election message if the id is lesser::two
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-2-1] INFO management - Received HB response from zero
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Leader Elected :: three
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Discard the election message if the id is lesser::two
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
[nioEventLoopGroup-2-1] INFO management - Received HB response from zero
[nioEventLoopGroup-5-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Leader Elected :: three
[inbound-mgmt-1] INFO management - Inbound Management Worker Called
```

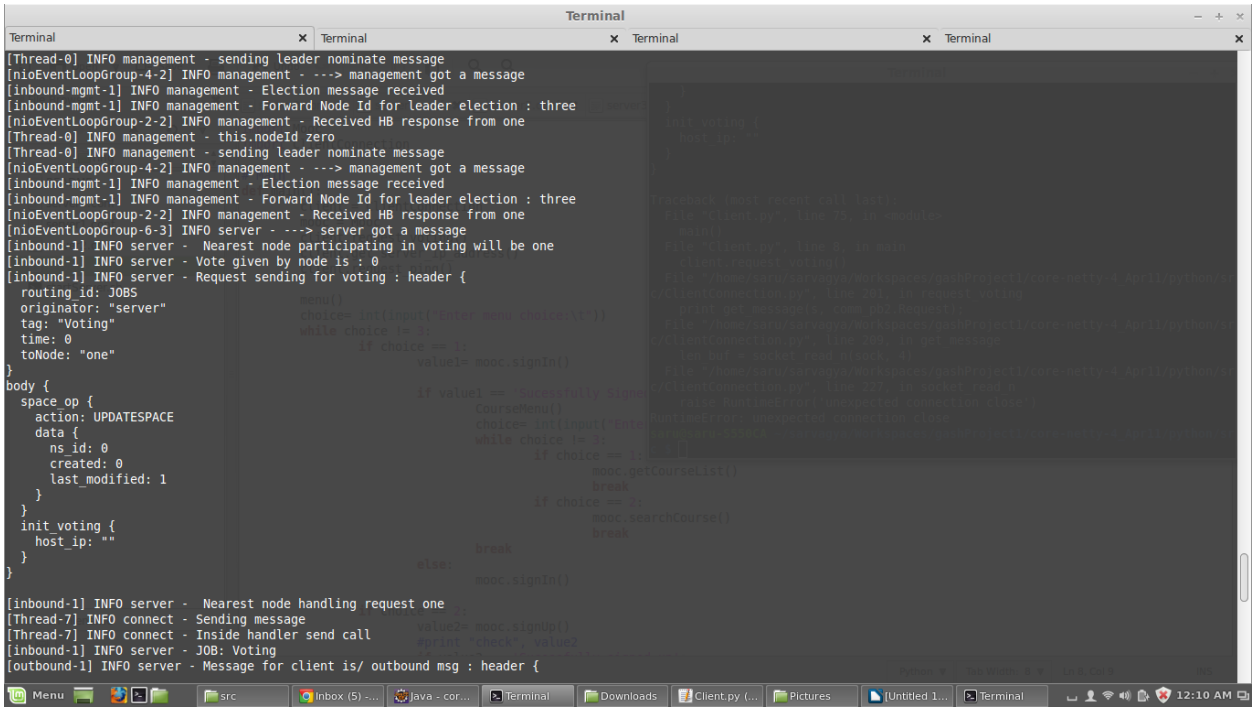
Riak Database Cluster Formation Screenshots

```
pipeline_create_one : 0
pipeline_create_error_count : 0
pipeline_create_error_one : 0
cpu_nprocs : 921
cpu_avg1 : 56
cpu_avg5 : 105
cpu_avg15 : 105
mem_total : 6140055552
mem_allocated : 5270204416
disk : [{"/",50730660,51},
        {"/sys/fs/cgroup",4,0},
        {"/dev",2983440,1},
        {"/run",599616,1},
        {"/run/lock",5120,0},
        {"/run/shm",2998072,1},
        {"/run/user",102400,1},
        {"/boot/efi",303104,15}]
nodename : 'riak@192.168.0.79'
connected_nodes : ['riak@192.168.0.71','riak@192.168.0.72']
sys_driver_version : <<"2.0">>
sys_global_heaps_size : 0
sys_heap_type : private
sys_logical_processors : 4
sys_otp_release : <<"R15B01">>
sys_process_count : 714
sys_smp_support : true
sys_system_version : <<"Erlang R15B01 (erts-5.9.1) [source] [64-bit] [smp:4:4] [async-threads:64] [kernel-poll:true]">>
sys_system_architecture : <<"x86_64-unknown-linux-gnu">>
sys_threads_enabled : true
sys_thread_pool_size : 64
sys_wordsize : 8
ring_members : ['riak@192.168.0.71','riak@192.168.0.72','riak@192.168.0.78',
                'riak@192.168.0.79']
ring_num_partitions : 64
ring_ownership : <<"[{'riak@192.168.0.71',16},\n {'riak@192.168.0.72',16},\n {'riak@192.168.0.78',24},\n {'riak@192.168.0.79',8}]">>
ring_creation_size : 64
storage_backend : riak_kv_bitcask_backend
erlydtl_version : <<"0.7.0">>
riak_control_version : <<"1.4.4-0-g9a74e57">>
cluster_info_version : <<"1.2.4">>
riak_search_version : <<"1.4.8-0-gbe6e4ed">>
merge_index_version : <<"1.3.2-0-gcb38ee7">>
```

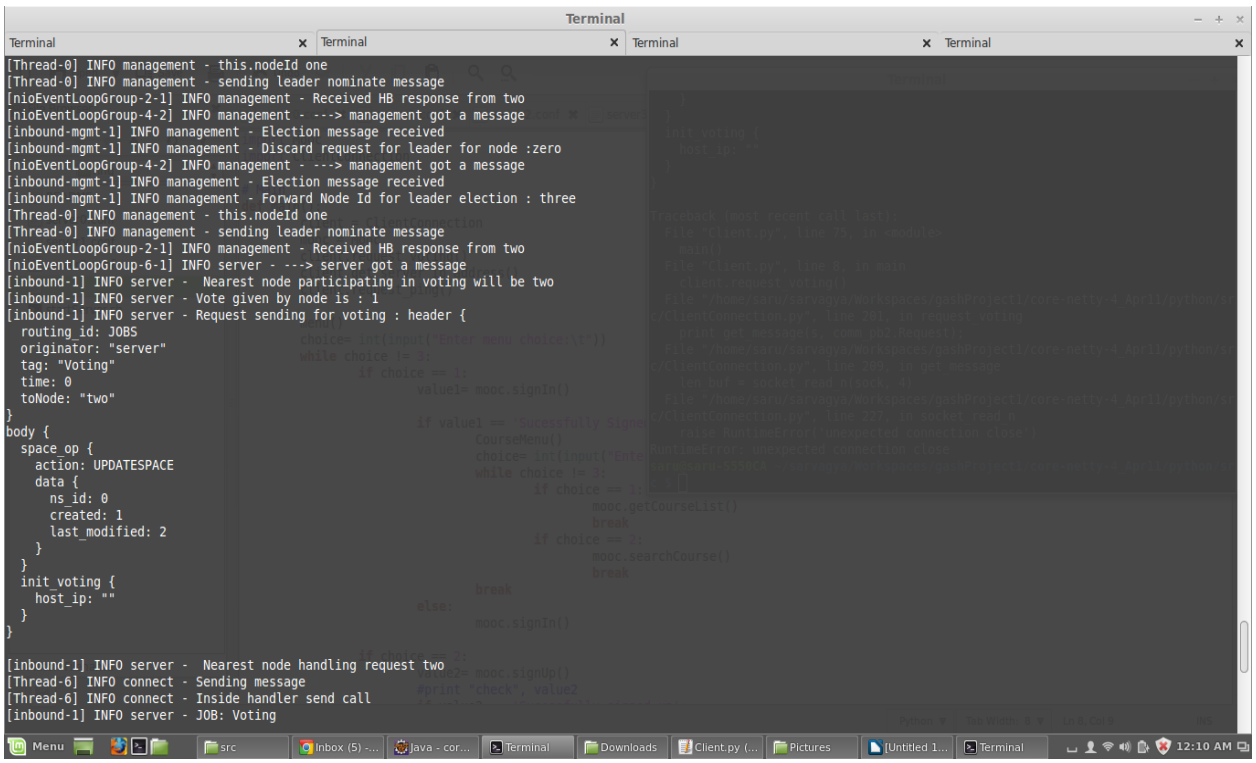
```
64 bytes from 192.168.0.78: icmp_seq=1 ttl=64 time=0.366 ms
64 bytes from 192.168.0.78: icmp_seq=2 ttl=64 time=0.366 ms
64 bytes from 192.168.0.78: icmp_seq=3 ttl=64 time=0.387 ms
^C
--- 192.168.0.78 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.366/0.373/0.387/0.009 ms
saru@saru-S550CA ~ $ sudo riak-admin cluster join riak@192.168.0.78
Success: staged join request for 'riak@192.168.0.79' to 'riak@192.168.0.78'
saru@saru-S550CA ~ $ sudo riak-admin cluster plan
============================== Staged Changes ==============================
Action         Details(s)
-------------------------------------------------------------------------------
join           'riak@192.168.0.79'
-------------------------------------------------------------------------------


NOTE: Applying these changes will result in 1 cluster transition

###############################################################################
                         After cluster transition 1/1
###############################################################################

================================ Membership ================================
Status     Ring    Pending    Node
-------------------------------------------------------------------------------
valid     100.0%     50.0%    'riak@192.168.0.78'
valid       0.0%     50.0%    'riak@192.168.0.79'
-------------------------------------------------------------------------------
Valid:2 / Leaving:0 / Exiting:0 / Joining:0 / Down:0

WARNING: Not all replicas will be on distinct nodes

Transfers resulting from cluster changes: 32
  32 transfers from 'riak@192.168.0.78' to 'riak@192.168.0.79'

saru@saru-S550CA ~ $ sudo riak-admin cluster commit
Cluster changes committed
saru@saru-S550CA ~ $ riak-admin status | grep riak_members
saru@saru-S550CA ~ $ riak-admin status | grep ring_members
ring_members : ['riak@192.168.0.78','riak@192.168.0.79']
saru@saru-S550CA ~ $
```

Voting Screenshots:

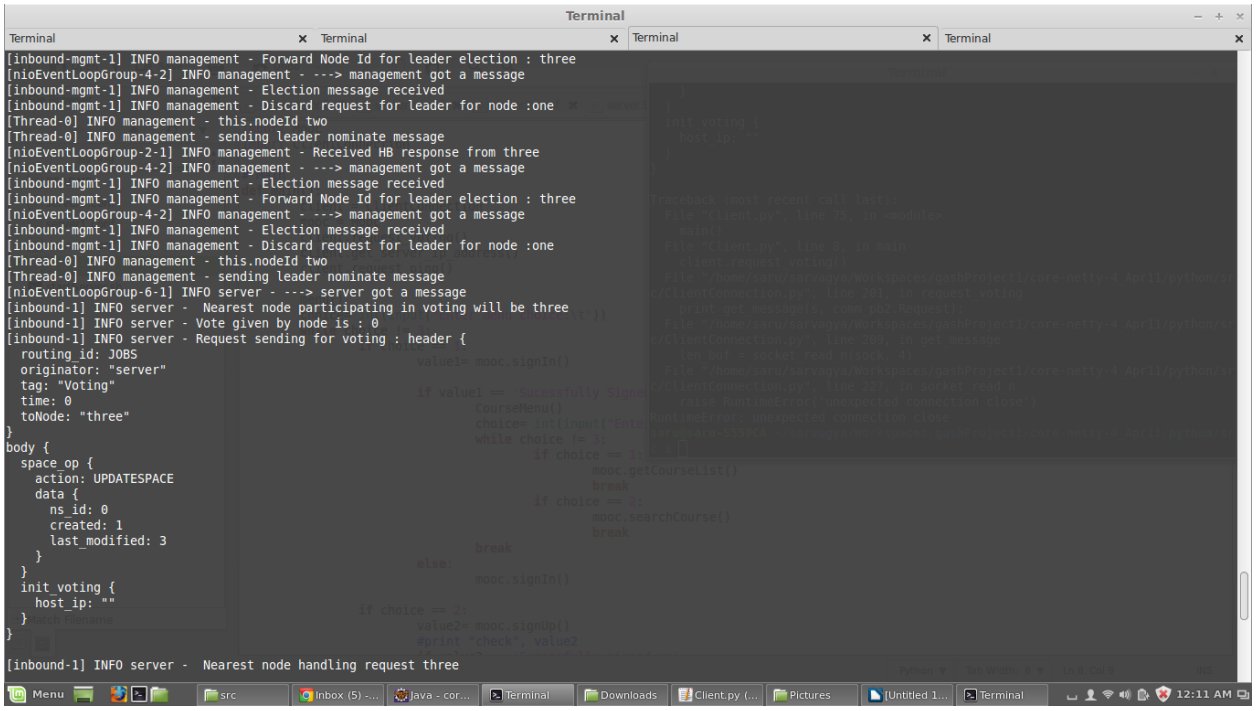
Server 0:



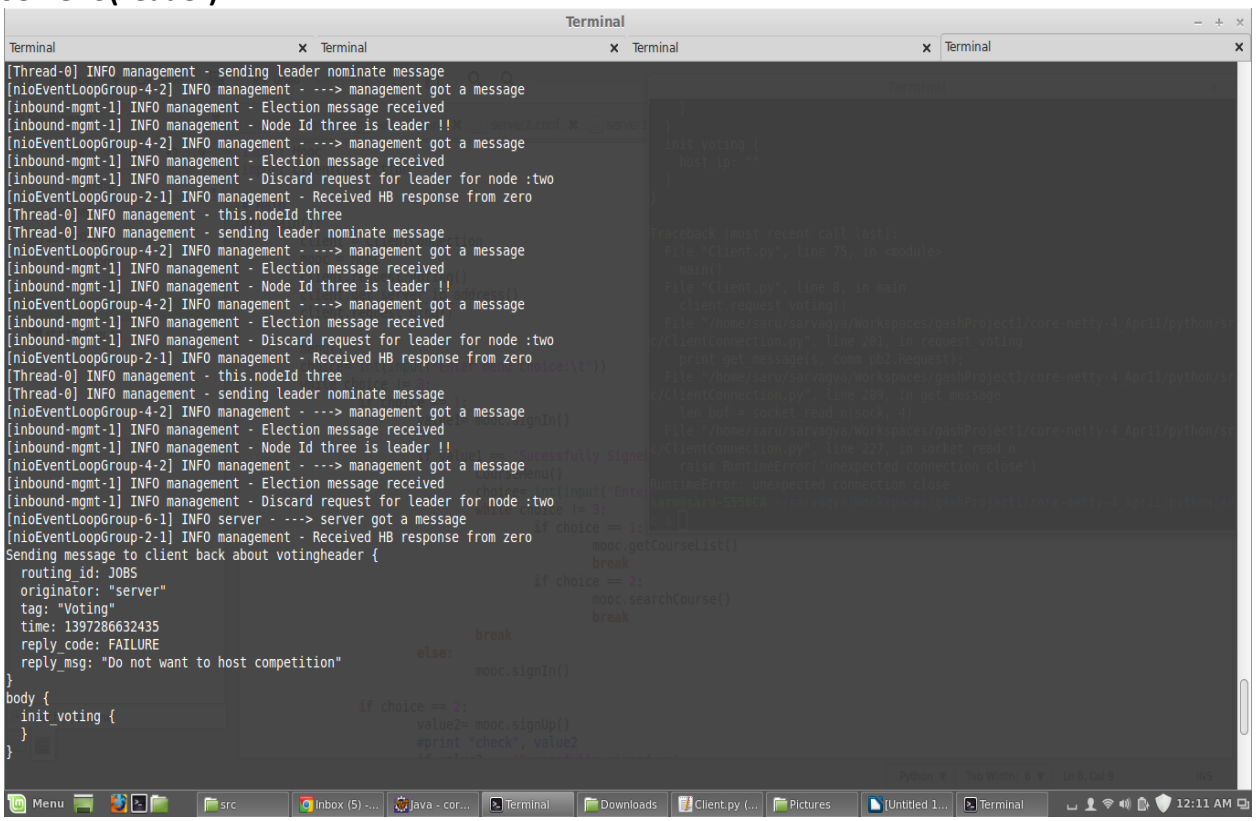
Server 1:


Terminal
[Thread-0] INFO management - this.nodeId one
[Thread-0] INFO management - sending leader nominate message
[nioEventLoopGroup-2-1] INFO management - Received HB response from two
[nioEventLoopGroup-4-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Election message received
[inbound-mgmt-1] INFO management - Discard request for leader for node :zero
[nioEventLoopGroup-4-2] INFO management - ---> management got a message
[inbound-mgmt-1] INFO management - Election message received
[inbound-mgmt-1] INFO management - Forward Node Id for leader election : three
[Thread-0] INFO management - this.nodeId one
[Thread-0] INFO management - sending leader nominate message
[nioEventLoopGroup-2-1] INFO management - Received HB response from two
[nioEventLoopGroup-6-1] INFO server - ---> server got a message
[inbound-1] INFO server - Nearest node participating in voting will be two
[inbound-1] INFO server - Vote given by node is : 1
[inbound-1] INFO server - Request sending for voting : header {
routing_id: JOBS
originator: "server"
tag: "Voting"
time: 0
toNode: "two"
}
body {
space_op {
action: UPDATESPACE
data {
ns_id: 0
created: 1
last_modified: 2
}
}
init_voting {
host_ip: ""
}
}
[inbound-1] INFO server - Nearest node handling request two
[Thread-6] INFO connect - Sending message
[Thread-6] INFO connect - Inside handler send call
[inbound-1] INFO server - JOB: Voting
Menu
src
Inbox (5) -...
Java - cor...
Terminal
Downloads
Client.py (...
Pictures
[Untitled 1...
Terminal
12:10 AM

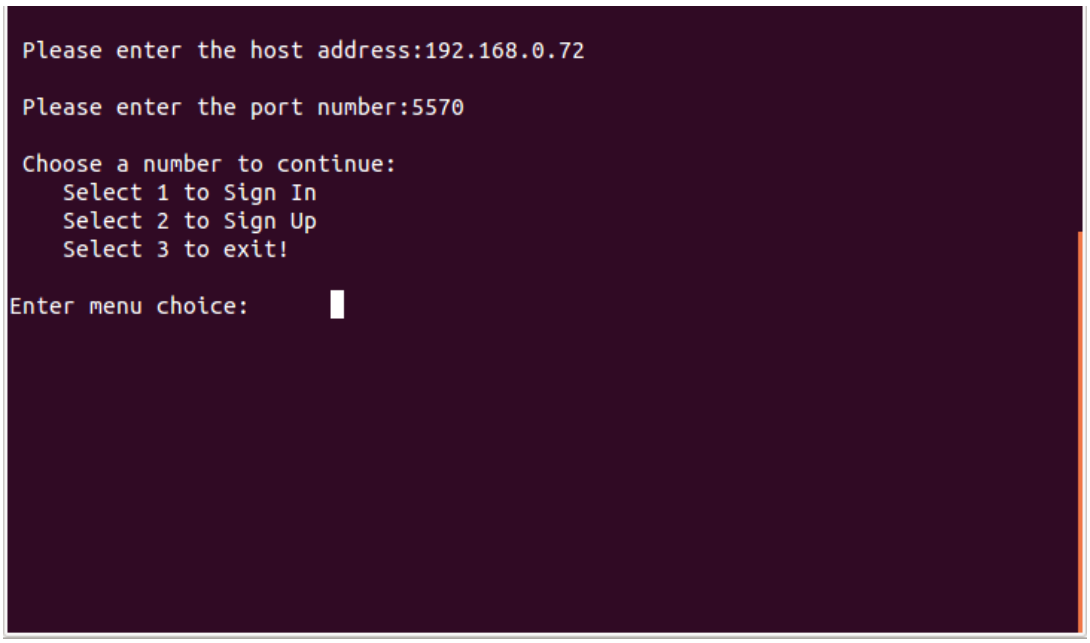
Server 2:



Server 3(Leader):



Client Screenshots:



References

1. https://www.found.no/foundation/leader-election-in-general/
2. http://en.wikipedia.org/wiki/Distributed_Systems#Coordinator_Election
3. http://seeallhearall.blogspot.com/2012/05/netty-tutorial-part-1-introduction-to.html
4. http://basho.com/riak/
5. http://en.wikipedia.org/wiki/Riak
6. https://developers.google.com/protocol-buffers/docs/javatutorial
7. https://www.found.no/foundation/leader-election-in-general/