CMPE275 PROJECT 1

Sharding Using a Mesh Topology

| Arun Malik | Ashley Love | Prasad Bidwai | Ryan Vo | Sai Karra
San Jose State University | Fall 2015

Table of Contents

List of Figures

List of Tables

LIST OF CODE SNIPPETS

Introduction

Project Overview

The focus of this project is to construct the backend for an asynchronous distributed photo storage system, called LifeForce. Ideally, the design shall not have a single point of failure and the client should be able to receive appropriate responses from any cluster in the network, given certain conditions. Each cluster in the network has a unique responsibility or implementation. In the instance that a node within a cluster cannot service the client's request, the request shall be forwarded to another cluster until the request is fulfilled. The aforementioned is the common interface for intercluster communication, which is represented in Figure 1.

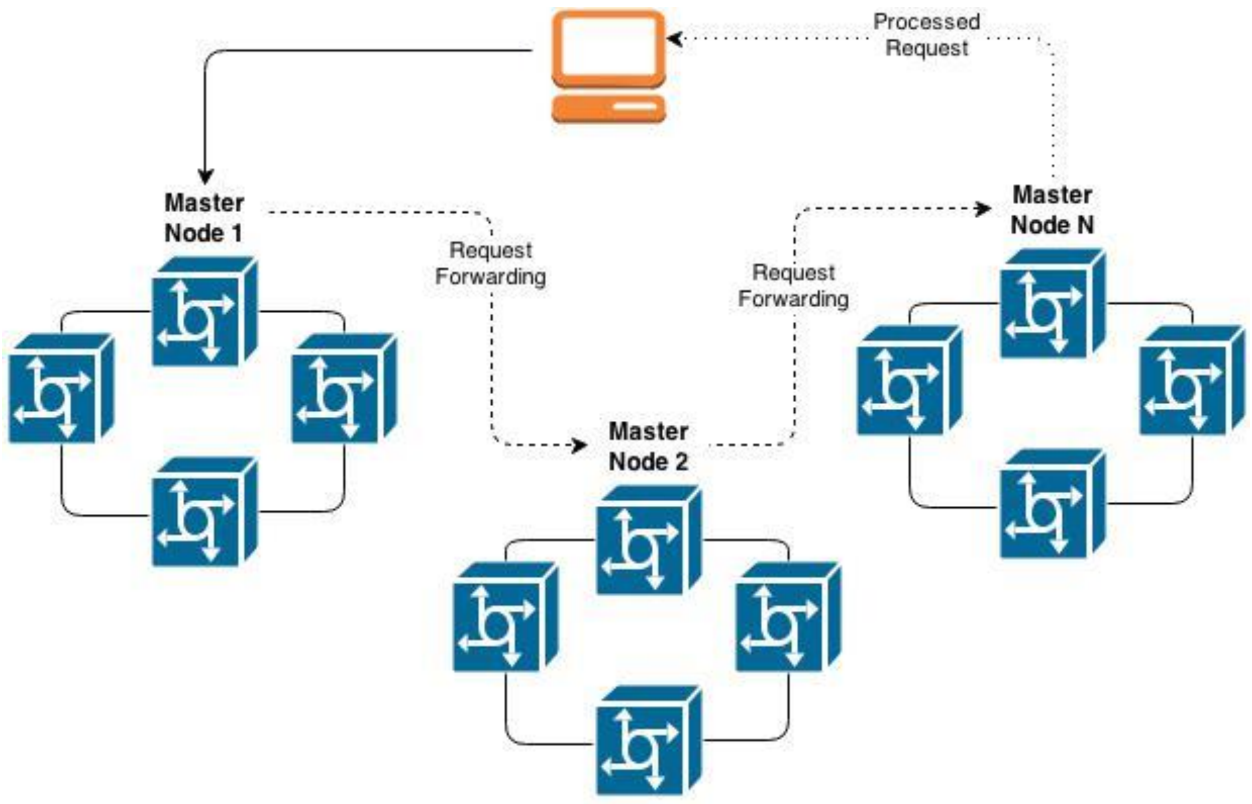


Figure 1. Common Interface Overview

The focal point of our intracluster communication, shall be sharding via a peerless network or mesh topology. This paper will investigate the implemented approaches, advantages and disadvantages, technical stack, and other concepts utilized to complete this project.

Problem Statement

The tech industry has grown from a monolithic architecture to a client-server model for managing data and requests. Regardless of these advances, there are still inherent problems that exist for quickly accessing, updating, and retrieving information, such as:

- Computational Cost: Concurrency, write, and update operations will require increased overhead [1].
- Database Consistency and Integrity: Modifications, such as write and update, will require significant management, which can lead to data inconsistency [1] [2].
- Complexity due to Distributed Nature: Distributed systems and databases require an additional layer of complexity, to manage information amongst nodes and clusters. Not accounting for the increased complexity and concurrency issues for distributed queries, can result in advantages, such as performance, becoming disadvantages. [2]
- Large Index and Table Size: Substantial amounts of data can be difficult to efficiently access and receive [2] [3].
- Degraded Performance: Sizable data sets can lead to decreased performance.

Proposed Solution

Due to new trends, such as DBaaS (Distributed Database as a Service) and extending approaches employed by the multibillion dollar industry, we propose to analyze and potentially mitigate the aforementioned problems by sharding. Using sharding and a mesh topology, a lightweight approach will be devised to decrease index size and computational expense, while increasing the performance. The system shall have: (1) an elected master node for communicating with the external network, (2) replication to eliminate SPF (Single Points of Failure), (3) a heartbeat to signify when a system goes down, (3) asynchronous communication, (5) and a replicated reverse proxy system to relay the master node to the client.

There will still be complexity for the distributed system due to sharding, but the advantages of the proposed sharding design outweighs the drawbacks of the status quo. The subsequent section shall discuss sharding, the architecture being implemented, and other technologies that shall work concurrently with the system.

Project Goals

The goal of this project is to create a distributed backend photo system that performs CRUD operations and provides the following constraints for inter and intracluster design:

- Scalablility (1)
- Asynchronous Communication(2)
- Reliability (3)
- Fault Tolerant (4)
- Lightweight (5)
- Low Latency (6)

Below are the corresponding features that shall be implemented to verify and validate the expected characteristics. The numbers above connect each characteristic to a corresponding feature.

Table 1. Inter and Intra-cluster Communication System Features and Characteristic Relation

Intercluster Communication	Intracluster Communication	Characteristic Satisfied
Client receives requests from a cluster's master node without knowing the cluster's topology or number of cluster's in the system	Identify if request can be handled within the cluster and handle or forward accordingly	(1)(2)(5)
Simulate different size systems by altering requests frequency	Handle DNS or flooded requests	(1)(2)(3)(6)
Handle exceptions (ie. Request cannot be serviced by any cluster, corrupted or incorrect data, etc)	Concurrency and asynchronous data handling, data replication, and exception handling	(2)(3)(4)(6)

Sharding

Sharding Overview

Sharding is the technique of distributing data across multiple servers to increase availability, performance, and scalability [3]. In layman's terms, sharding can be explicated by the idea's name. A broken piece of glass, or a "shard", is a fragmented piece necessary to completely recreate or construct the glass, which is how databases use the abstraction. Sharding by function, master key, or some other method, amalgamates or retrieves the necessary shards for the needed snapshot. This concept is great for Big Data or high transaction systems because sharding allows large amounts of data to be more effectively manipulated. Historically, sharding is more straightforward using a NoSQL database because SQL will need to have data split across multiple tables (an un-organic method), whereas NoSQL can have XML fragments that are easier to distribute [3]. In spite of this fact, MySQL is implemented and the reasoning is discussed in 'Table 6. Time required to transmit data 10,000 times

In addition, another benefit of Protobuf is that it allows for cross-language communication, unlike serialization methods such as Java Serialization. In our project, our Python client was able to talk to the Java server using Protobuf. Although Protobuf is officially supported in 3 languages, this is few compared to other serialization methods like Thrift, which supports at least a dozen languages.

Protobuf provides a way to serialize/deserialize data to send across the wire. According to Igor Aninschenko, compared to many other ways to serialize data, such as JSON, Protobuf is one of the lightest and fastest. He found that when he serialized some sample data using Protobuf, it needed 250 bytes to serialize, while JSON needed 559 bytes, XML 836, and Thrift 278. When he transmitted the data 10,000 times, Protobuf took 1 min 19 s, while JSON took 4 min 45 s, XML 5 min 27 s, and Thrift 1 min 5 s. Protobuf was quick. In addition, unlike serialization methods such as Java Serialization, Protobuf allows for cross-language communication. In our project, our Python client talked to the Java server using Protobuf. Although Protobuf is officially supported in 3 languages, other serialization methods, such as Thrift, are supported in many more.

SQL' section. This "shared-nothing" architecture was made mainstream by Google's BigTable, but has been formerly been labeled as horizontal partitioning [4]. Sharding is a growing abstraction that is used in many industries from websites, complex analytics, and gaming engines, such as Amazon, Facebook, Youtube and more. The popularity and necessity to have a sharding solution has led to widespread support and integration of sharding. (ie. IBM Informix or ElasticSearch). The '**Horizontal Sharding'** section contains the bulk of information and analysis detail, since the final code implementation is a form of horizontal sharding.

Advantages and Disadvantages of Sharding

There are many advantages and disadvantages for sharding. Below is a high level overview describing the pros and cons. Though Table 2, gives an initial impression that sharding is always the solution, this is not true. Due to the significant amount of complexity needed, technical constraints, workload, and work type, there are extenuating ramifications that can skew the scale.

Table 2. Advantages and Disadvantages of Sharding

Advantages	Disadvantages
Parallel Processing	Complexity
Scalability	Single Point of Failure
Availability	Complex Logic Needed for SQL
Simplified Working Set	Cross-shard Joins
Reliability	Auto Increment
Management and Maintenance	Frequent Writes
Performance	Memory Needed for Large Working Sets
Cost	Operational Complexity
Distributed Queries	-
Mixed Workloads	-
Big Data Applications	-
Highly Transactional Applications	-

Select characteristics of sharding and the relevance will be discussed in the following subsections.

Performance

Sharding will allow increased performance because the shards can be processed in parallel in the distributed homogenous architecture being utilized. When accessing or updating data frequently, the data can be retrieved more expeditiously when grouped based on a key or function that creates a natural grouping or data access pattern. The data set being used will be much smaller, hence the significance of performance gains may not be as significant.

Scalability, Reliability, and Availability

Sharding, along with Netty's asynchronous NIO API, makes the design very scalable. Typical resource constraints, such as CPU, disk, or other bottlenecks, overcomes these issues because of dispersed resources and smaller data sets. A single node does not have to bear the brunt of the processing and computation when sharding. Instead, the work is dispersed, which allows systems to work optimally because of the small data sets and asynchronous calls. Reliability is one of the down falls of sharding, hence replication shall be implemented as a workaround, as show in Figure 1 below.

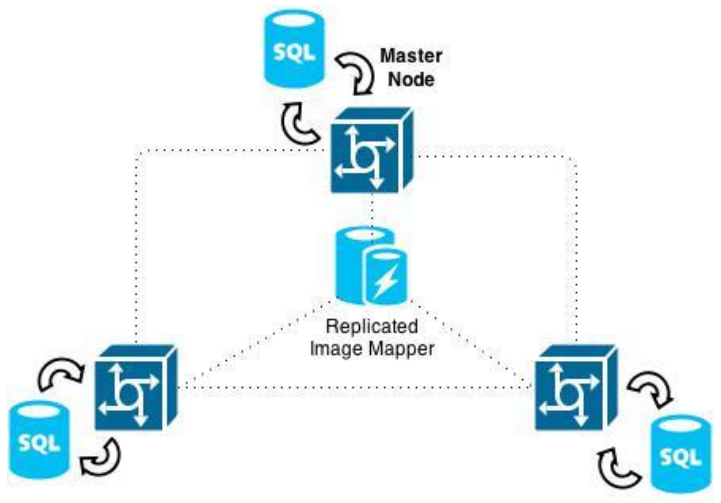


Figure 2. Mesh Design with Replication

Cross-Shard Joins and Auto Increment

Cross shard joins and auto increments are important factors to consider when selecting sharding as an option. In future project versions this would be more of a design constraint. Currently, the system is using a UUID created by a Java API and cross-shard joins will not be addressed.

Sharding and Approach Justification

Vertical Sharding

Vertical sharding is a technique used to abridge the number of columns in order to reduce I/O operations and increase performance [5]. This approach is not ideal for the LifeForce project because of the nominal amount of metadata included. For example, the initial design includes only a handful of columns and concentrates on performing CRUD operations, therefore vertical sharding would only increase the system complexity. For future versions, that may include more columns, such as 'caption', 'comments', 'number of clicks', number of shares', and so forth, vertical partitioning will be a more suitable and viable option.

Horizontal Sharding

A form of horizontal partitioning was the original intention of this project. For the type of data and range of potential requests that the system may handle or receive, horizontal sharding is a well-suited solution. At the embryonic stage, the plan was for a heterogeneous P2P network to process client requests and group partitions based on username or date, then update a replicated mapper table that correlates 'image UUID' and 'node id.' Due to the lean metadata, username and date were ideal shard keys because:

- No other information will be provided, either by the client or node, except 'blob storage id', 'uuid,' 'image,' 'time,' 'mapper id,' and 'node id,' which limits the sharding key options, as a result of the proto file and intercluster communication design 'contract' (hence username was later discarded as a plausible sharding key option).
- Photos frequently being accessed by upload date is a logical schema.

Initial Shard Design

The phase 1 developed architecture, was as follows:

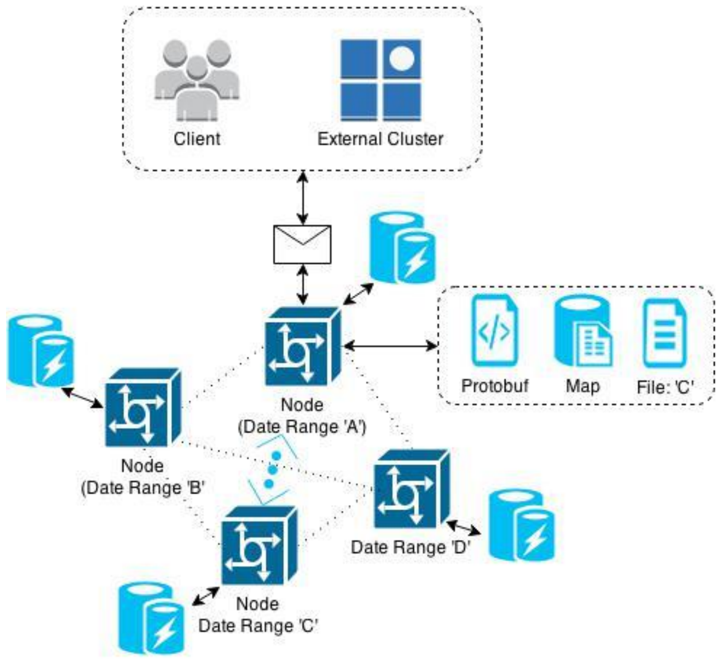


Table 3. Date Based Sharding Key Architecture

In the above, the message is deserialized using Protobuf and categorized into a date range that corresponds to a specific shard. The dotted rectangle represents that the message can either be coming directly from the client or from an external cluster. The intercommunication illustrated can establish contact with any node elected as the master because of the P2P design. A master, or the idea of a master, only exists to conform to the class standard for communication amongst networks

Final Shard Design

Due to time constraints and full functionality concerns, a modified round robin shard design was implemented:

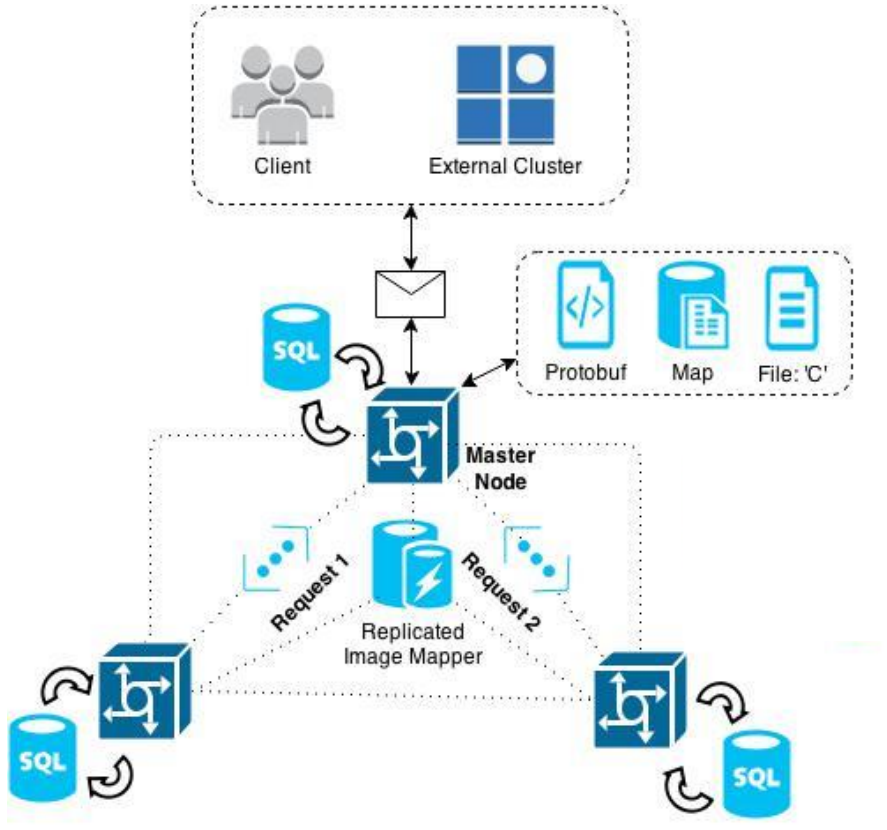


Figure 3. Round Robin Design

This system acts as a round robin to handle write requests. **Whenever a node fails,** the request shall be automatically forwarded to the next available node. The dashed rectangle describing contents of Node 0 are present in all the nodes because each can act as the master. In the instance of a read request, the elected master shall look at the 'mapper' to find the node containing the image. The design shall be able to have N number of nodes. Only three nodes are shown in **Figure 3** for simplicity. For a complete system representation, see the

Topology section.

Technical Stack and Project Tools

The technical stack for this project involves the following:

Table 4. Project Tools

Tool	Technology
Language	Java and Python
Storage	MySql
Core Package	Jboss and NIO
Data Serialization	Google Protobuf
IDE	Eclipse and PyCharm
Operating System	Ubuntu 14.1, Mac 10.10, Windows 8

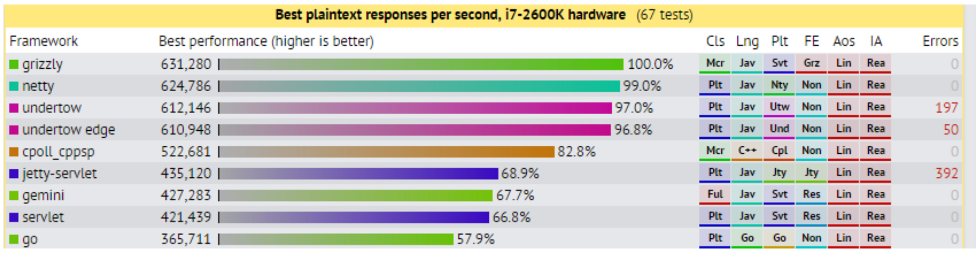
For each of the components in the technical stack, the advantages, disadvantages, reasoning, and comparison to other potential technology selections shall be discussed.

Netty

Netty is a project constraint, but also a beneficial tool for server and socket implementation. The primary advantage is that it is a non-blocking event-driven asynchronous I/O client-server application framework. Asynchronous communication enables simplicity, scalability, flexibility, and decouples the one-to-one relationship of sockets to requests. Non-blocking I/O allows Netty to responds quickly and without having to wait of other process or requests to complete and was designed to avoid the limitations in popular protocols, such as FTP, SMTP, HTTP, and other legacy protocols.

Advantages

Netty is designed to be a high-performance server framework managing to handle 624,786 plaintext responses per second in a relatively recent benchmarking and whose performance is only outperformed by Java's Project Grizzly, another NIO-based application framework built on top of the native Java NIO framework, [6] (see figure on the right). Despite its obvious performance goals, Netty is not isolated to a single protocol or codec but supports protocols: TCP, UDP, UDT, SCTP and codecs: HTTP 1.0/2.0, SPDY, SSL/TLS, WebSockets, DNS, Stomp, Protocol Buffers, and



Best plaintext responses per second, i7-2600K hardware (67 tests)

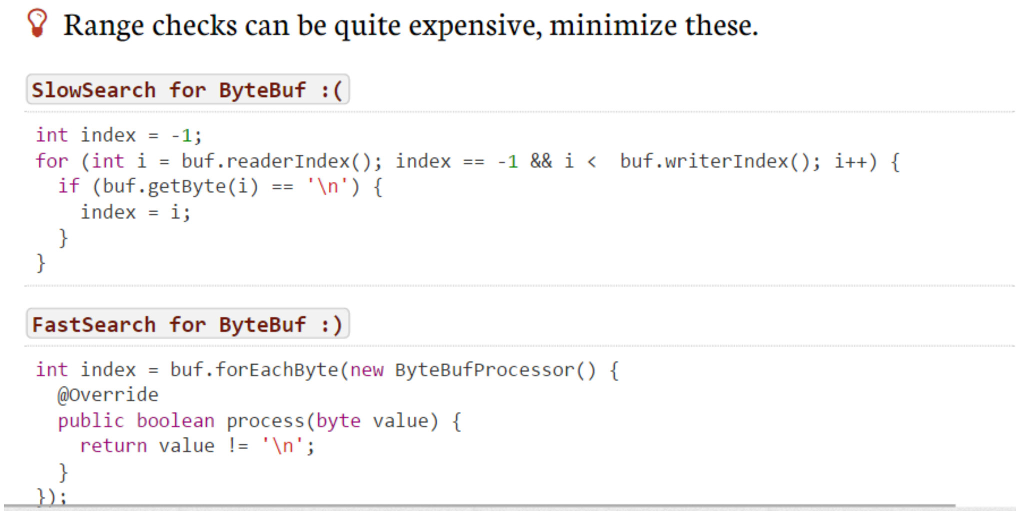
Framework	Best performance (higher is better)		Cls	Lng	Plt	FE	Aos	IA	Errors
grizzly	631,280	100.0%	Mcr	Jav	Svt	Grz	Lin	Rea	0
netty	624,786	99.0%	Plt	Jav	Nty	Non	Lin	Rea	0
undertow	612,146	97.0%	Plt	Jav	Utw	Non	Lin	Rea	197
undertow edge	610,948	96.8%	Plt	Jav	Und	Non	Lin	Rea	50
cpoll_cppsp	522,681	82.8%	Mcr	C++	Cpl	Non	Lin	Rea	0
jetty-servlet	435,120	68.9%	Plt	Jav	Jty	Jty	Lin	Rea	392
gemini	427,283	67.7%	Ful	Jav	Svt	Res	Lin	Rea	0
servlet	421,439	66.8%	Plt	Jav	Svt	Res	Lin	Rea	0
go	365,711	57.9%	Plt	Go	Go	Non	Lin	Rea	0

Figure 4. Tech Empower's Framework benchmarking results

many more [7]. To add to the diversity, the consistent and modular API and the ability to chain procedure to create a process allows developers to develop a custom codec or protocol to address the needs of their specific environment and goals [8]. Netty also uses native applications such as OpenSSL and system calls, to speed up the process of certain processes, such as communication encryption and file I/O.

Disadvantages

One of the most important goals of Netty is performance but there are certain limitations to the performance. While the framework itself performs well as the benchmarking results above show, the application of a developer must meet the stringent programming frameworks of the library. For example, a developer may opt for the use of a for-loop, a simple strategy to read a channel. However, this strategy will cause slow performance in the framework as detailed in the figure to the right [7]. The lack of insight into the framework will not provide the advertised performance out-of-the-box. Another performance concern to be noted is that while an application can be optimized heavily, it is still at the mercy of the programming language that the application runs on top of. Java is an extremely capable language that has great performance to a certain extent. The JVM can be given more memory or moved to a system that can operate at faster frequencies but the JVM cannot utilize more than a single process. The JVM can only scale up not out, which will become more and more expensive to handle the growing needs. One interesting aspect of this framework is that while Netty is quite capable, there is a larger and more popular NIO asynchronous event-driven framework called Node.js. JavaScript seems to be the next big thing as it's attracting more and more developers and enterprises, only adding to the popularity of Node.js over Netty.


Range checks can be quite expensive, minimize these.

SlowSearch for ByteBuf :(

```
int index = -1;
for (int i = buf.readerIndex(); index == -1 && i <  buf.writerIndex(); i++) {
  if (buf.getByte(i) == '\n') {
    index = i;
  }
}
```

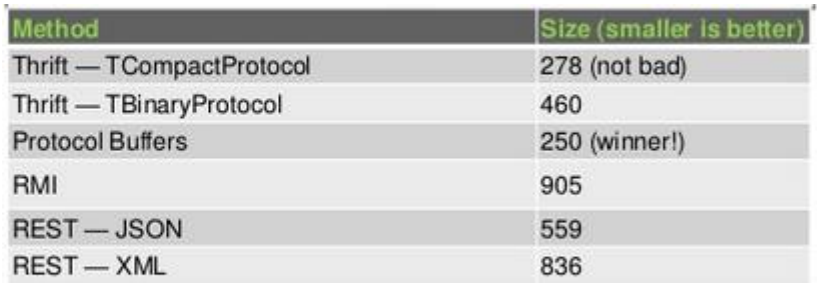
FastSearch for ByteBuf :)

```
int index = buf.forEachByte(new ByteBufProcessor() {
  @Override
  public boolean process(byte value) {
    return value != '\n';
  }
});
```

Figure 5. Netty optimizations

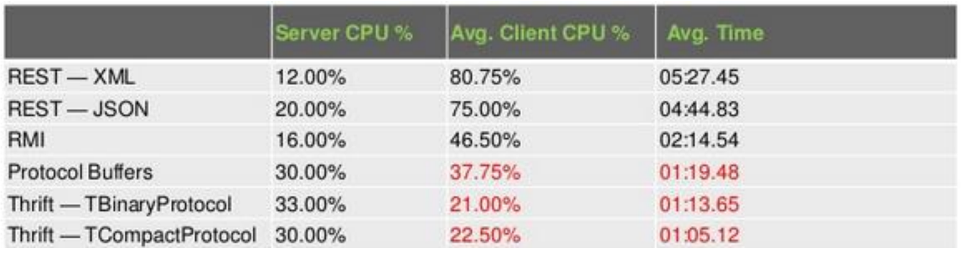
Protobuf

Protobuf provides a way to serialize or deserialize data to send across the wire. According to Igor Aninschenko, compared to many other ways to serialize data, such as JSON, Protobuf is one of the lightest and fastest [9]. He found that when he serialized data using Protobuf, only 250 bytes was needed to encode the data, while JSON needed 559 bytes, XML 836, and Thrift 278 (using the TCompactProtocol). When he transmitted the data 10,000 times, Protobuf took 1 min 19 s, while JSON took 4 min 45 s, XML 5 min 27 s, and Thrift 1 min 5 s. Hence, validating Protobuf's light and quick specifi.



Method	Size (smaller is better)
Thrift — TCompactProtocol	278 (not bad)
Thrift — TBinaryProtocol	460
Protocol Buffers	250 (winner!)
RMI	905
REST — JSON	559
REST — XML	836

Table 5. Size required to encode data



	Server CPU %	Avg. Client CPU %	Avg. Time
REST — XML	12.00%	80.75%	05:27.45
REST — JSON	20.00%	75.00%	04:44.83
RMI	16.00%	46.50%	02:14.54
Protocol Buffers	30.00%	37.75%	01:19.48
Thrift — TBinaryProtocol	33.00%	21.00%	01:13.65
Thrift — TCompactProtocol	30.00%	22.50%	01:05.12

Table 6. Time required to transmit data 10,000 times

In addition, another benefit of Protobuf is that it allows for cross-language communication, unlike serialization methods such as Java Serialization. In our project, our Python client was able to talk to the Java server using Protobuf. Although Protobuf is officially supported in 3 languages, this is few compared to other serialization methods like Thrift, which supports at least a dozen languages.

Protobuf provides a way to serialize/deserialize data to send across the wire. According to Igor Aninschenko, compared to many other ways to serialize data, such as JSON, Protobuf is one of the lightest and fastest. He found that when he serialized some sample data using Protobuf, it needed 250 bytes to serialize, while JSON needed 559 bytes, XML 836, and Thrift 278. When he transmitted the data 10,000 times, Protobuf took 1 min 19 s, while JSON took 4 min 45 s, XML 5 min 27 s, and Thrift 1 min 5 s. Protobuf was quick. In addition, unlike serialization methods such as Java Serialization, Protobuf allows for cross-language communication. In our project, our Python client talked to the Java server using Protobuf.

Although Protobuf is officially supported in 3 languages, other serialization methods, such as Thrift, are supported in many more.

SQL

MySQL was selected as the tool for storing and retrieving client requests. A relational database offers more reliability due to the ACID nature, but NoSQL is more ideal for scalability and performance. For extremely large databases, the structured nature of MySQL will require joins, hashing, and other functionality that needs customized code, which will lead to network latency. Excessive atypical data calls and transformations will yield a pseudo-MySQL that cannot garnish the ingrained system optimizations, efficiency, and other underlying framework that makes MySQL a wholesome tool [4] [10]. For the quick uptime, amount of data being stored, and number of requests that were being handled, MySQL is sufficient enough to satisfy design and performance constraints. In the long run, the system should be implemented with NoSQL or a combination of the two database types.

The following command line output shows the DB schema:

```
mysql> select blobStorageId, uuid, caption, createdBy, createdDate, lastModifiedBy,
lastModifiedDate from blobStorage;

+---------------+----------------+-----------+-------------+---------------+------------------+
| blobStorageId | uuid | caption | createdBy | createdDate | lastModifiedBy | lastModifiedDate |
| 43 | 1d1440e6-d3c6-41ec-8356-93276601a831 | varun   | NULL      | NULL | NULL | NULL     |
| 44 | 0864bbff-3616-441c-b70c-14e94513ca3b | picture | NULL      | NULL | NULL | NULL     |
| 45 | 9427df5f-fc1f-461c-ad8f-6a9ed3f3f10a | input_image.jpg | NULL | NULL | NULL  | NULL |
+---------------+-------------------------------------+-----------------+-----------+----------
3 rows in set (0.00 sec)


mysql> select * from mapper;
+----------+--------+--------------------------------------+
| idmapper | nodeid | uuid                                 |
+----------+--------+--------------------------------------+
|        2 |      0 | 1d1440e6-d3c6-41ec-8356-93276601a831 |
|        3 |      0 | 0864bbff-3616-441c-b70c-14e94513ca3b |
|        4 |      1 | f3d0db31-9eae-48c3-9e3c-d85bbefa3ec5 |
|        5 |      0 | 9427df5f-fc1f-461c-ad8f-6a9ed3f3f10a |
+----------+--------+--------------------------------------+
4 rows in set (0.00 sec)
```

Code Snippet 1. Command Line Representation of DB Schema

Since replication was not the design focal point, replication is implemented using the 'MySQL Sandbox' tool. There were complications for having multiple MySQL instances on the same machine. This sandbox tool allowed multiple instances to be installed without having to worry about privileges or managing ports. More information about the tool can be found at mysqlsandbox.net.

System Design and Architecture

Server Architecture

The server side architecture completes requests to read and write images, which accesses or updates images accordingly. The network has a peerless design, hence when the client requests a CRUD operation, any one of the peers in our network can answer the request completely and independently. The Netty package, NIO, is used to manage the asynchronous calls made between the server and the client. For instance, listeners are instantiated that "listen" for the asynchronous return call and activate a set of responses on the call back. An example, is displayed below:

```
public StartElection(String ip, int port) {

        MgmtClientCommand cc = new MgmtClientCommand(ip, port);
        MgmtListener listener = new MgmtPrintListener("demo");

        cc.addListener(listener);
        cc.startElection(ip, port);
}
```

Code Snippet 2. Asynchronous

`StartElection()` has a listener, `'MgmtListener'`, which is queued until a specific event occurs, a change in the network mandating a new election. The client manager, `'MgmtClientCommand'`, has connected listeners, which causes a series of client commands and an election once the listener obtains the correct "event." Having a fault tolerant election is essential for intercluster communication.

In the case of server or node failure, several scenarios can happen: (1) for a failed write request, the algorithm is constructed to skip the failed node and write to the next node's database and (2) for read requests, the data shall be retrieved from the read replica.

Server Architecture Influences

In Peer-to-Peer Overlays: Structure, Unstructured or Both, the paper surveys the advantages and disadvantages of structured and unstructured overlay networks. The fundamental concepts in the paper aided in the construction of the LifeForce P2P unstructured overlay system.

What is an unstructured overlay network?

An unstructured overlay network has nodes that form connections as needed without any permanent coupling between any nodes [11]. Peers can enter and leave the network with minimal overhead especially during situations of high churn [11] [12]. An unstructured design scales well, but for our specific needs, the churn "should" be minimal since the system will have a maximum of 5 nodes, which is a design decision. The goal is to have all 5 nodes working with minimal fluctuation with regards to the number of functional nodes, hence this benefit for unstructured nodes was nominal. The attractiveness stems from the ability to quickly build a network with reasonably low complexity and high fault tolerance. The following is an example of a mesh or unstructured overlay architecture:

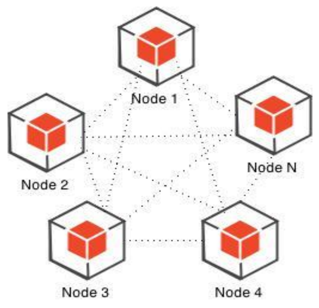


Figure 6. Unstructured Overlay Architecture

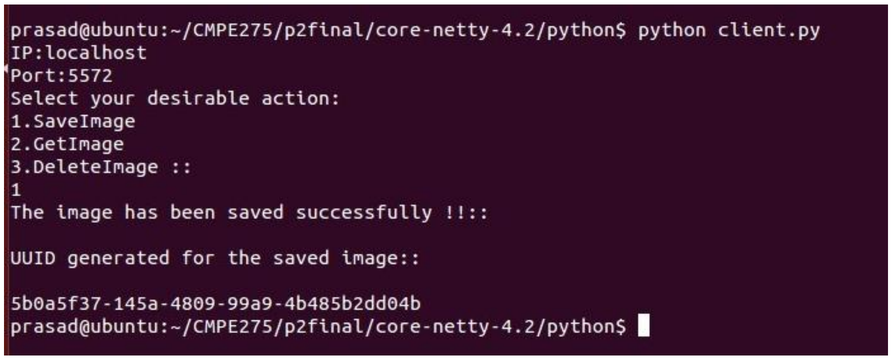
Client Side Architecture

Python Client

The Python client is used to store, retrieve, or delete an image. The user is prompted to enter an IP address and port of the server machine. Next, the operation to be performed on an image is selected: save, retrieve, or delete. After saving an image, the user is returned a UUID generated for the image. For retrieving or deleting an image, the user will need to provide the image UUID. The saved image is read into binary format then encoded using base64 encoding technique before being serialized with Google Protobuf.

Below are screen captures to show the client's interface:

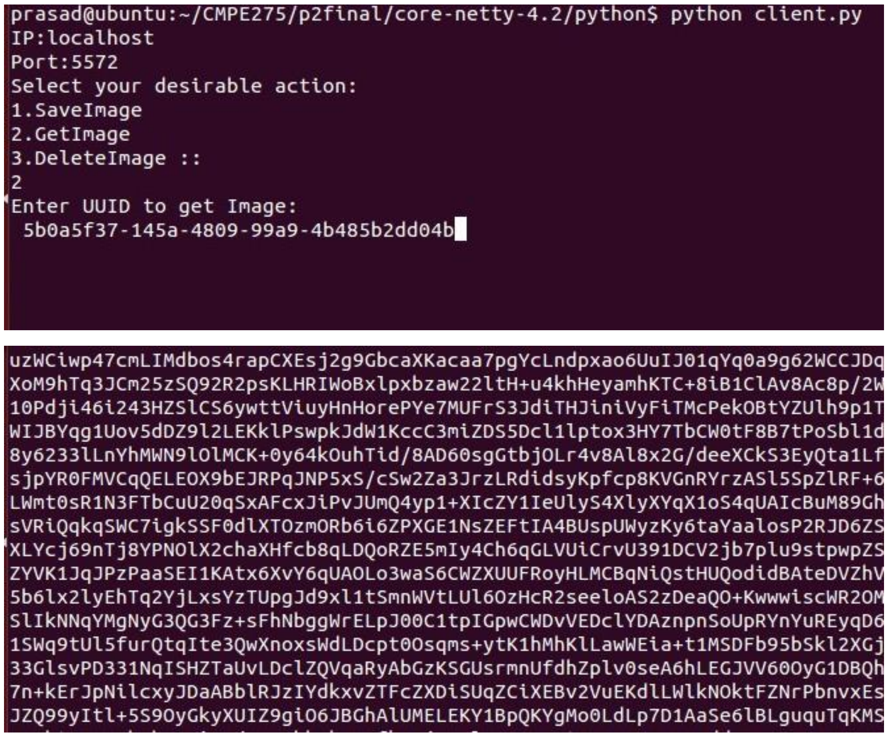
Save Image operation



```
prasad@ubuntu:~/CMPE275/p2final/core-netty-4.2/python$ python client.py
IP:localhost
Port:5572
Select your desirable action:
1.SaveImage
2.GetImage
3.DeleteImage ::
1
The image has been saved successfully !!::

UUID generated for the saved image::

5b0a5f37-145a-4809-99a9-4b485b2dd04b
prasad@ubuntu:~/CMPE275/p2final/core-netty-4.2/python$
```

Figure 7. Client Interface for Save Image Operation

Retrieve Image operation

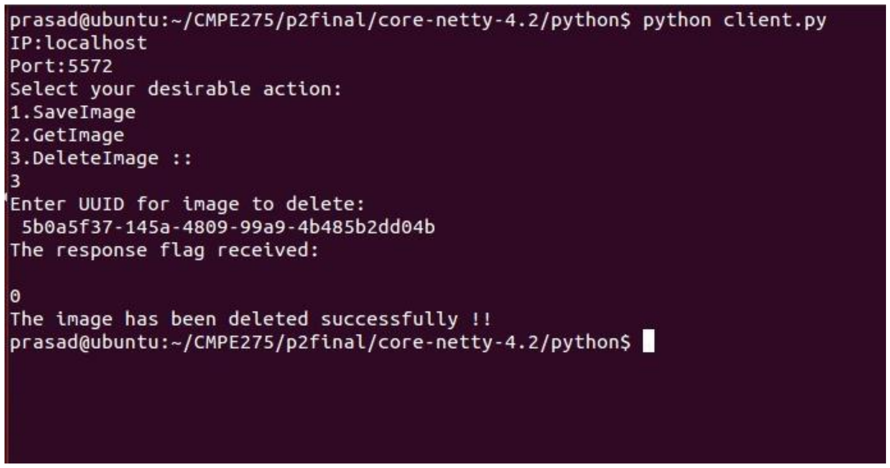


```
prasad@ubuntu:~/CMPE275/p2final/core-netty-4.2/python$ python client.py
IP:localhost
Port:5572
Select your desirable action:
1.SaveImage
2.GetImage
3.DeleteImage ::
2
Enter UUID to get Image:
 5b0a5f37-145a-4809-99a9-4b485b2dd04b
```

```
uzWCiwp47cmLIMdbos4rapCXEsj2g9GbcaXKacaa7pgYcLndpxao6UuIJ01qYq0a9g62WCCJDq
XoM9hTq3JCm25zSQ92R2psKLHRIWoBxlpxbzaw22ltH+u4khHeyamhKTC+8iB1ClAv8Ac8p/2W
10Pdji46i243HZSlCS6ywttViuyHnHorePYe7MUFrS3JdiTHJiniVyFiTMcPekOBtYZUlh9p1T
WIJBYqg1Uov5dDZ9l2LEKklPswpkJdW1KccC3miZDS5Dcl1lptox3HY7TbCW0tF8B7tPoSbl1d
8y6233lLnYhMWN9lOlMCK+0y64kOuhTid/8AD60sgGtbjOLr4v8Al8x2G/deeXCkS3EyQta1Lf
sjpYR0FMVCqQELEOX9bEJRPqJNP5xS/cSw2Za3JrzLRdidsyKpfcp8KVGnRYrzASl5SpZlRF+6
LWmt0sR1N3FTbCuU20qSxAFcxJiPvJUmQ4yp1+XIcZY1IeUlyS4XlyXYqX1oS4qUAIcBuM89Gh
sVRiQqkqSWC7igkSSF0dlXTOzmORb6i6ZPXGE1NsZEFtIA4BUspUWyzKy6taYaalosP2RJD6ZS
XLYcj69nTj8YPNOlX2chaXHfcb8qLDQoRZE5mIy4Ch6qGLVUiCrvU391DCV2jb7plu9stpwpZS
ZYVK1JqJPzPaaSEI1KAtx6XvY6qUAOLo3waS6CWZXUUFRoyHLMCBqNiQstHUQodidBAteDVZhV
5b6lx2lyEhTq2YjLxsYzTUpgJd9xl1tSmnWVtLUl6OzHcR2seeloAS2zDeaQO+KwwwiscWR2OM
SlIkNNqYMgNyG3QG3Fz+sFhNbggWrELpJ00C1tpIGpwCWDvVEDclYDAznpnSoUpRYnYuREyqD6
1SWq9tUl5furQtqIte3QwXnoxsWdLDcpt0Osqms+ytK1hMhKlLawWEia+t1MSDFb95bSkl2XGj
33GlsvPD331NqISHZTaUvLDclZQVqaRyAbGzKSGUsrmnUfdhZplv0seA6hLEGJVV60yG1DBQh
7n+kErJpNilcxyJDaABblRJzIYdkxvZTFcZXDiSUqZCiXEBv2VuEKdlLWlkNOktFZNrPbnvxEs
JZQ99yItl+5S9OyGkyXUIZ9giO6JBGhAlUMELEKY1BpQKYgMo0LdLp7D1AaSe6lBLguquTqKMS
```

Figure 8. Client Interface for the binary data received during retrieve operation

Delete Image Operation:



```
prasad@ubuntu:~/CMPE275/p2final/core-netty-4.2/python$ python client.py
IP:localhost
Port:5572
Select your desirable action:
1.SaveImage
2.GetImage
3.DeleteImage ::
3
Enter UUID for image to delete:
 5b0a5f37-145a-4809-99a9-4b485b2dd04b
The response flag received:

0
The image has been deleted successfully !!
prasad@ubuntu:~/CMPE275/p2final/core-netty-4.2/python$
```

Figure 9. Client interface for deleting an image

Topology

The following figure is an illustration of the complete system architecture:

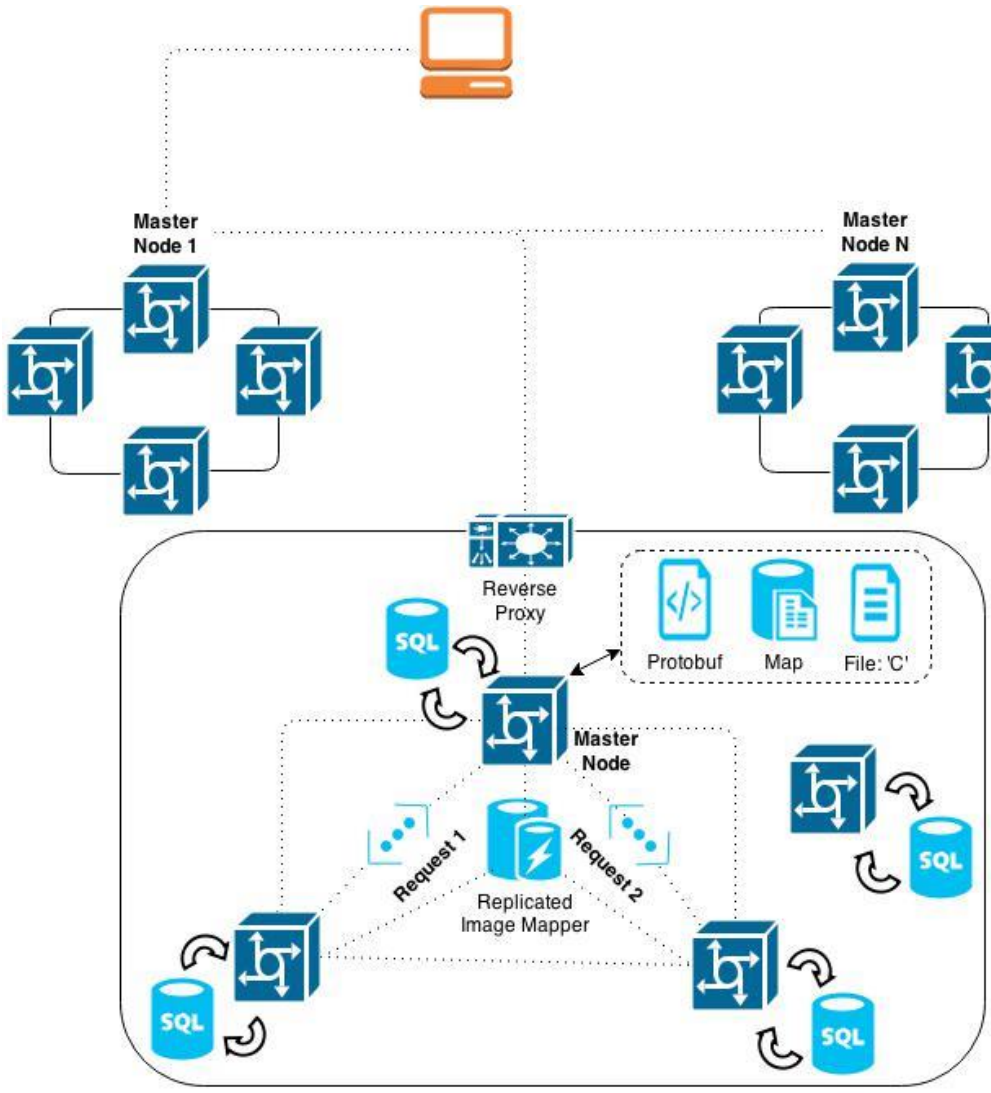


Figure 10. Topology Overview

The work flow for executing CRUD operations using the above topology is as follows:

Table 7. Workflow Description

WRITE / UPDATE / DELETE :	**READ :**
Client sends request to Leader	Client sends request to Leader
Leader initiates round-robin and send request to respective node for update (ie. Save, Modify, Delete)	Leader searches for UUID in mapping DB and retrieves respective nodeID
Node will respond with UUID of the image to leader	Leader sends request to respective node
Master node will update DB along with mapping for UUID and mapper correlation	Node sends response to leader
	Leader sends response to client

All of the subcomponents, such as sharding, netty, and so forth, culminate to create the topology above and logic for the workflow. The system constructed satisfies the basic project requirements and goals. Relevant code snippets and performance will be discussed in the following sections.

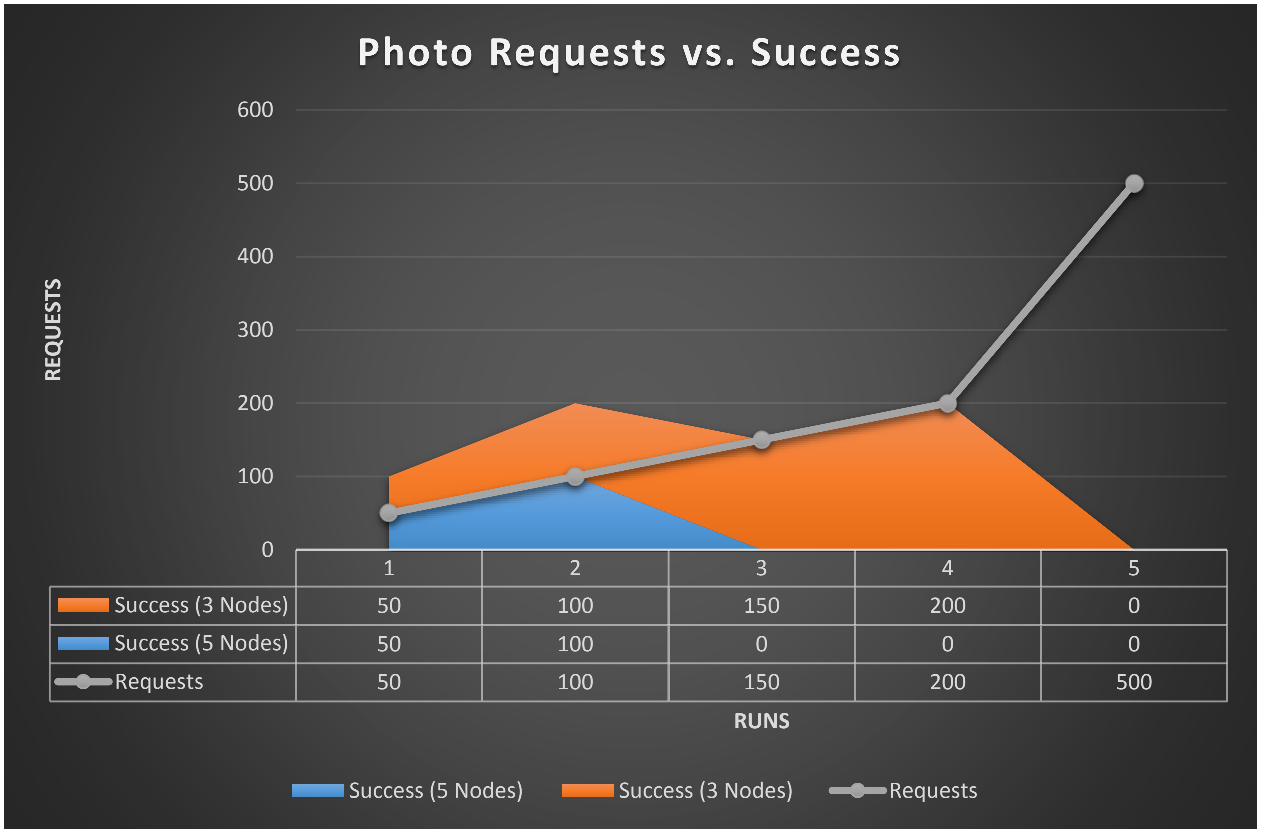
Reverse Proxy

As a single point of entry for the clients to the cluster, a reverse proxy was created. Whenever a client connects to the reverse proxy, the proxy creates a tunnel from the client to the node designated to handle client traffic. There is only one designated client traffic handler node. Sometimes, the designated node goes down, and when that happens the other nodes of the cluster elect a new designated client traffic handler. During this period, traffic is lost as the nodes hold an election. Once the nodes are done voting, they inform the reverse proxy of the decision through a management port, from which point on the reverse proxy will tunnel client traffic to the newly designated node.

System Performance: Test Day

When the developed system was integrated with the class, we were successfully able to test under a variety of conditions. Below is a graphical representations of the results:

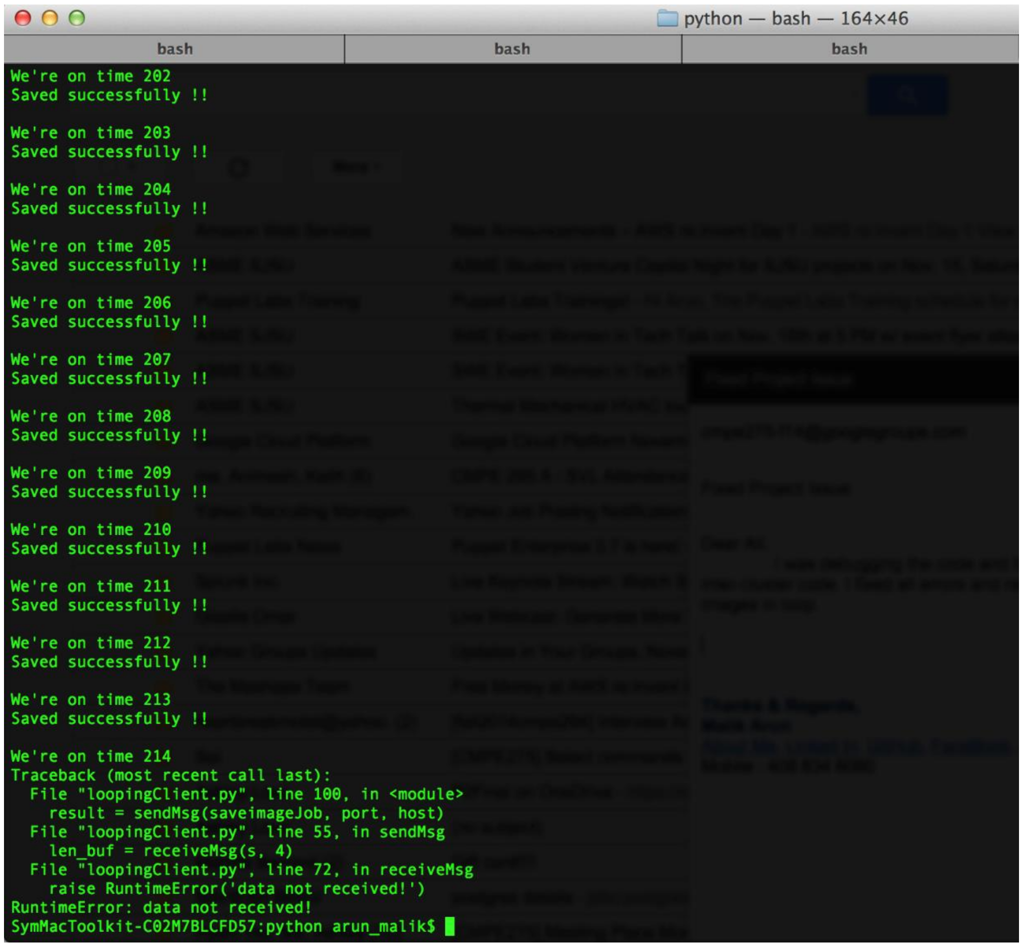
Test Day Results



	1	2	3	4	5
🟧 Success (3 Nodes)	50	100	150	200	0
🟦 Success (5 Nodes)	50	100	0	0	0
● Requests	50	100	150	200	500

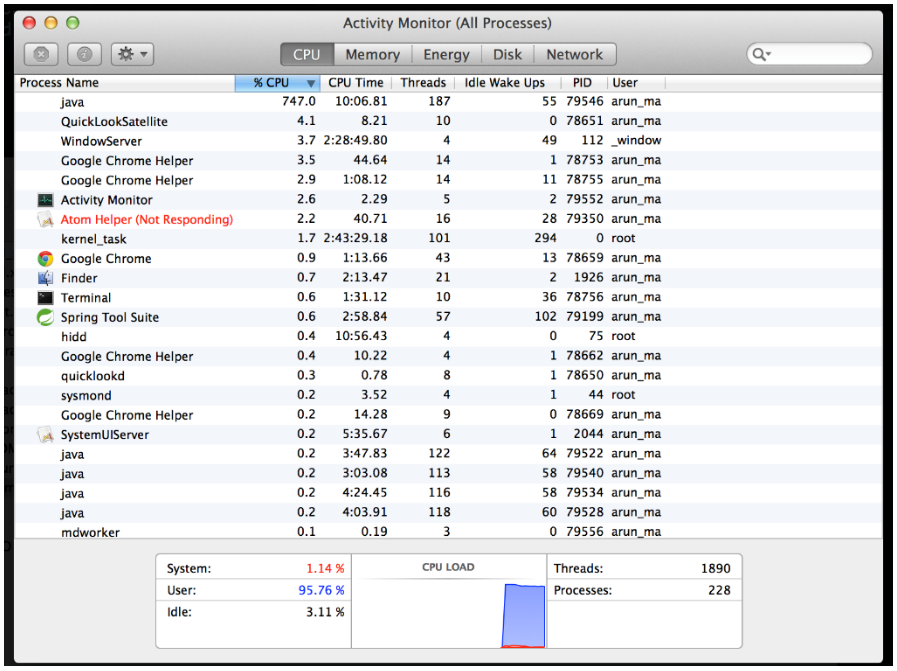
Figure 11. Photo Requests vs. Successes for Each Node Scenario
Note 1: *Microsoft Surface, Macbook Pro (x2), HP, and Lenovo systems were used*
Note 2: Microsoft Surface, 1 Macbook Pro, and Lenovo were the 3 stable systems

In the above, '0' indicates that the request could not be completed. If the system can only serve a partial number of requests before the next milestone, the lower milestone is kept. For example, for a marker of 500, if only 214 were executed, the graph will just show as milestone 200 met and milestone 500 not met. The above figure measures whether or not certain milestones are met. A later graph will show the max requests serviceable by 3, 4, or 5 node requests respectively. The below image is a sample scenario of the max number of requests serviced. From the CPU utilization, the system has serviced the maximum number of requests (>90% CPU utilization) and any more processing would result in severely diminished performance.



```
We're on time 202
Saved successfully !!

We're on time 203
Saved successfully !!

We're on time 204
Saved successfully !!

We're on time 205
Saved successfully !!

We're on time 206
Saved successfully !!

We're on time 207
Saved successfully !!

We're on time 208
Saved successfully !!

We're on time 209
Saved successfully !!

We're on time 210
Saved successfully !!

We're on time 211
Saved successfully !!

We're on time 212
Saved successfully !!

We're on time 213
Saved successfully !!

We're on time 214
Traceback (most recent call last):
  File "loopingClient.py", line 100, in <module>
    result = sendMsg(saveimageJob, port, host)
  File "loopingClient.py", line 55, in sendMsg
    len_buf = receiveMsg(s, 4)
  File "loopingClient.py", line 72, in receiveMsg
    raise RuntimeError('data not received!')
RuntimeError: data not received!
SymMacToolkit-C02M7BLCFD57:python arun_malik$
```

Figure 12. Cluster Job Processing Results

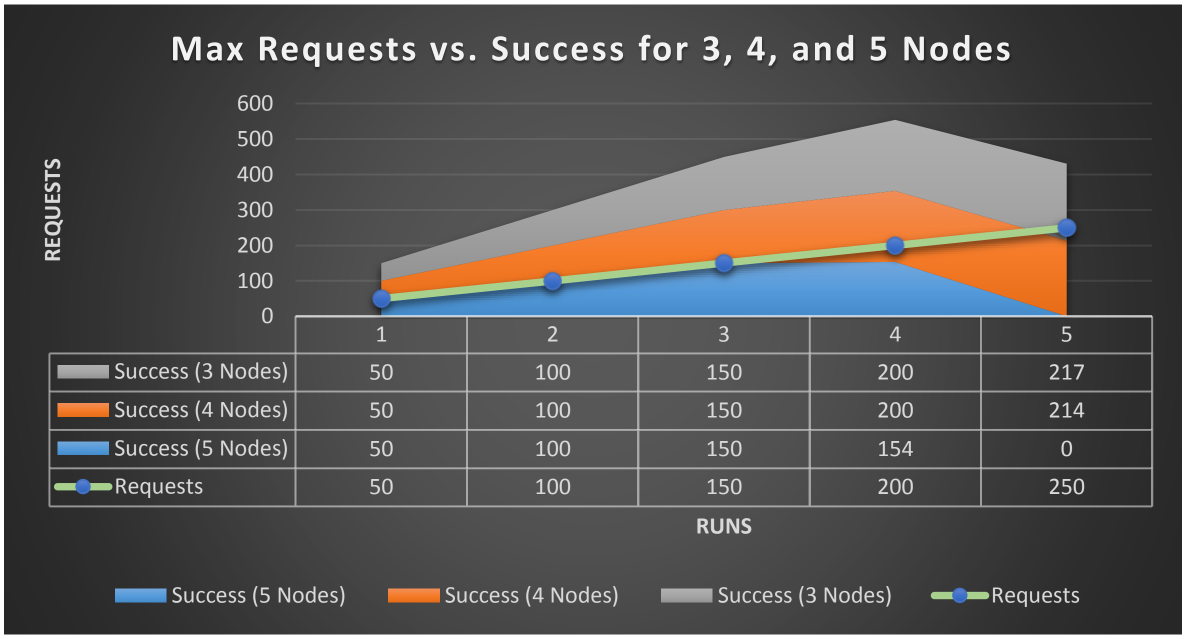


Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	PID	User
java	747.0	10:06.81	187	55	79546	arun_ma
QuickLookSatellite	4.1	8.21	10	0	78651	arun_ma
WindowServer	3.7	2:28:49.80	4	49	112	_window
Google Chrome Helper	3.5	44.64	14	1	78753	arun_ma
Google Chrome Helper	2.9	1:08.12	14	11	78755	arun_ma
Activity Monitor	2.6	2.29	5	2	79552	arun_ma
Atom Helper (Not Responding)	2.2	40.71	16	28	79350	arun_ma
kernel_task	1.7	2:43:29.18	101	294	0	root
Google Chrome	0.9	1:13.66	43	13	78659	arun_ma
Finder	0.7	2:13.47	21	2	1926	arun_ma
Terminal	0.6	1:31.12	10	36	78756	arun_ma
Spring Tool Suite	0.6	2:58.84	57	102	79199	arun_ma
hidd	0.4	10:56.43	4	0	75	root
Google Chrome Helper	0.4	10.22	4	1	78662	arun_ma
quicklookd	0.3	0.78	8	1	78650	arun_ma
sysmond	0.2	3.52	4	1	44	root
Google Chrome Helper	0.2	14.28	9	0	78669	arun_ma
SystemUIServer	0.2	5:35.67	6	1	2044	arun_ma
java	0.2	3:47.83	122	64	79522	arun_ma
java	0.2	3:03.08	113	58	79540	arun_ma
java	0.2	4:24.45	116	58	79534	arun_ma
java	0.2	4:03.91	118	60	79528	arun_ma
mdworker	0.1	0.19	3	0	79556	arun_ma

System:	1.14 %	Threads:	1890
User:	95.76 %	Processes:	228
Idle:	3.11 %		

Figure 13. Sample CPU Utilization Screenshot

Afterwards, the system was run for 3, 4, and 5 nodes to see what the maximum values were before memory swapping occurred:



	1	2	3	4	5
Success (3 Nodes)	50	100	150	200	217
Success (4 Nodes)	50	100	150	200	214
Success (5 Nodes)	50	100	150	154	0
Requests	50	100	150	200	250

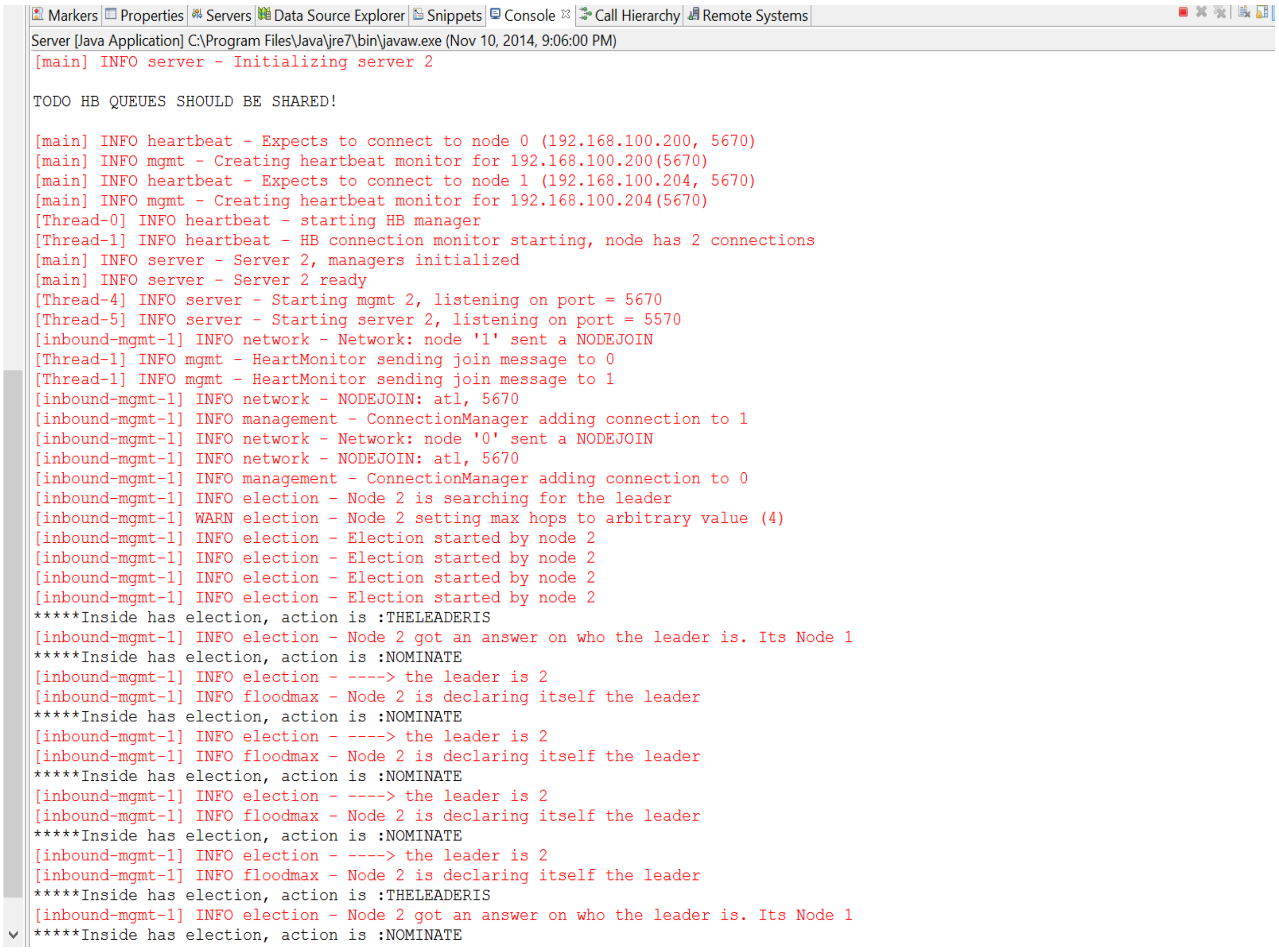
Figure 14.Max Requests vs. Success for 3, 4, and 5 Nodes

Ironically, more requests can be handled with fewer nodes, which may be due to the heterogeneity of the systems. More tests would have been completed, but there was an issue of stability for a 5 node system. Later on we decided to test using only 3 nodes using the three stable systems. Two computers had trouble with 'Out of Memory Exceptions' or were dropped from the network. With more testing using different scenarios, these problems can be avoided, such as tests using: (1) heterogeneous systems by configuring identical VMs and (2) modifying settings and configurations for Nodes with problems. Also, DNS attacks should have been handled more effectively by implementing a more thorough exception handler for system attacks and data floods.

Another problem was the election process. Whenever a new node joined the cluster, an election was re-initiated and the master had to be reelected, which caused problems. Hindsight, the ideal approach would be to have a re-election only when the master node goes down, but update the system and nodes to relay that the system has been modified. We also should have spent more time focusing on edge cases, simulated environments with varying nodes, and higher transaction volume.

Test Cases and Screenshots

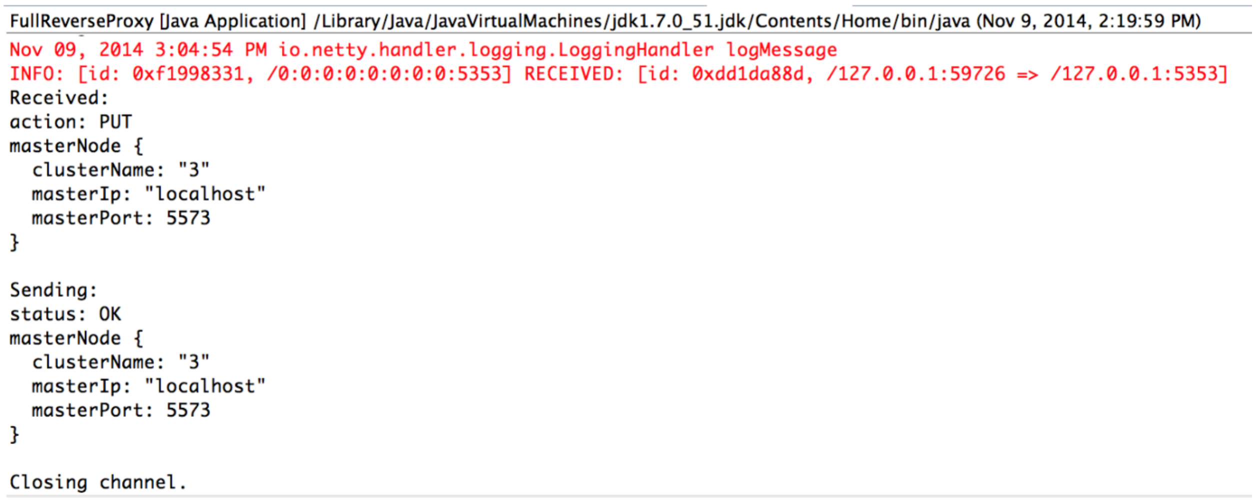
Test Case 1: Server Initialization and Election



```
Server [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 10, 2014, 9:06:00 PM)
[main] INFO server - Initializing server 2

TODO HB QUEUES SHOULD BE SHARED!

[main] INFO heartbeat - Expects to connect to node 0 (192.168.100.200, 5670)
[main] INFO mgmt - Creating heartbeat monitor for 192.168.100.200(5670)
[main] INFO heartbeat - Expects to connect to node 1 (192.168.100.204, 5670)
[main] INFO mgmt - Creating heartbeat monitor for 192.168.100.204(5670)
[Thread-0] INFO heartbeat - starting HB manager
[Thread-1] INFO heartbeat - HB connection monitor starting, node has 2 connections
[main] INFO server - Server 2, managers initialized
[main] INFO server - Server 2 ready
[Thread-4] INFO server - Starting mgmt 2, listening on port = 5670
[Thread-5] INFO server - Starting server 2, listening on port = 5570
[inbound-mgmt-1] INFO network - Network: node '1' sent a NODEJOIN
[Thread-1] INFO mgmt - HeartMonitor sending join message to 0
[Thread-1] INFO mgmt - HeartMonitor sending join message to 1
[inbound-mgmt-1] INFO network - NODEJOIN: atl, 5670
[inbound-mgmt-1] INFO management - ConnectionManager adding connection to 1
[inbound-mgmt-1] INFO network - Network: node '0' sent a NODEJOIN
[inbound-mgmt-1] INFO network - NODEJOIN: atl, 5670
[inbound-mgmt-1] INFO management - ConnectionManager adding connection to 0
[inbound-mgmt-1] INFO election - Node 2 is searching for the leader
[inbound-mgmt-1] WARN election - Node 2 setting max hops to arbitrary value (4)
[inbound-mgmt-1] INFO election - Election started by node 2
[inbound-mgmt-1] INFO election - Election started by node 2
[inbound-mgmt-1] INFO election - Election started by node 2
[inbound-mgmt-1] INFO election - Election started by node 2
*****Inside has election, action is :THELEADERIS
[inbound-mgmt-1] INFO election - Node 2 got an answer on who the leader is. Its Node 1
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :THELEADERIS
[inbound-mgmt-1] INFO election - Node 2 got an answer on who the leader is. Its Node 1
*****Inside has election, action is :NOMINATE
```

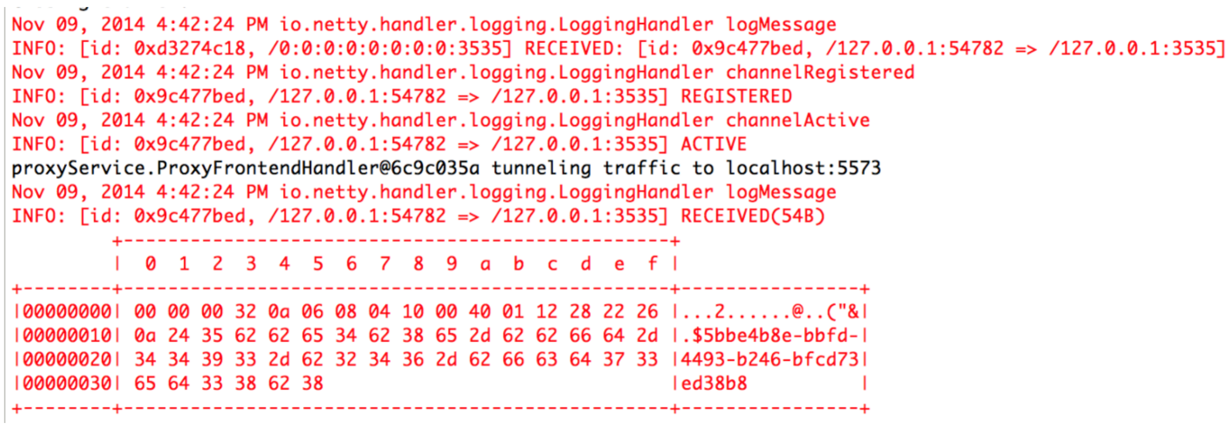
Figure 15. Server Initialization and Node Election for Master Node to Participate in Intercluster Communication

Test Case 2: Reverse Proxy



```
FullReverseProxy [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_51.jdk/Contents/Home/bin/java (Nov 9, 2014, 2:19:59 PM)
Nov 09, 2014 3:04:54 PM io.netty.handler.logging.LoggingHandler logMessage
INFO: [id: 0xf1998331, /0:0:0:0:0:0:0:0:5353] RECEIVED: [id: 0xdd1da88d, /127.0.0.1:59726 => /127.0.0.1:5353]
Received:
action: PUT
masterNode {
  clusterName: "3"
  masterIp: "localhost"
  masterPort: 5573
}

Sending:
status: OK
masterNode {
  clusterName: "3"
  masterIp: "localhost"
  masterPort: 5573
}

Closing channel.
```

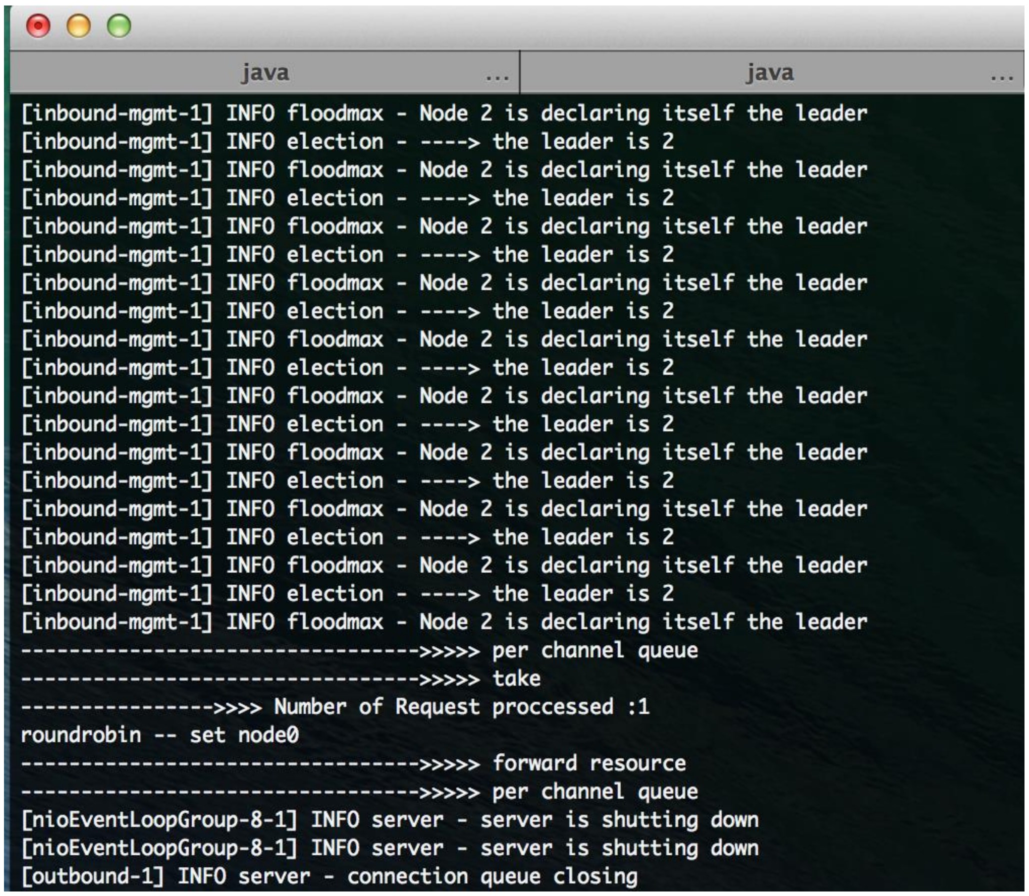
Figure 16. Reverse proxy receives request on Management Port that tunnels traffic to localhost:5573.



```
Nov 09, 2014 4:42:24 PM io.netty.handler.logging.LoggingHandler logMessage
INFO: [id: 0xd3274c18, /0:0:0:0:0:0:0:0:3535] RECEIVED: [id: 0x9c477bed, /127.0.0.1:54782 => /127.0.0.1:3535]
Nov 09, 2014 4:42:24 PM io.netty.handler.logging.LoggingHandler channelRegistered
INFO: [id: 0x9c477bed, /127.0.0.1:54782 => /127.0.0.1:3535] REGISTERED
Nov 09, 2014 4:42:24 PM io.netty.handler.logging.LoggingHandler channelActive
INFO: [id: 0x9c477bed, /127.0.0.1:54782 => /127.0.0.1:3535] ACTIVE
proxyService.ProxyFrontendHandler@6c9c035a tunneling traffic to localhost:5573
Nov 09, 2014 4:42:24 PM io.netty.handler.logging.LoggingHandler logMessage
INFO: [id: 0x9c477bed, /127.0.0.1:54782 => /127.0.0.1:3535] RECEIVED(54B)
         +-------------------------------------------------+
         |  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f |
+--------+-------------------------------------------------+----------------+
|00000000| 00 00 00 32 0a 06 08 04 10 00 40 01 12 28 22 26 |...2......@..("&|
|00000010| 0a 24 35 62 62 65 34 62 38 65 2d 62 62 66 64 2d |.$5bbe4b8e-bbfd-|
|00000020| 34 34 39 33 2d 62 32 34 36 2d 62 66 63 64 37 33 |4493-b246-bfcd73|
|00000030| 65 64 33 38 62 38                               |ed38b8          |
+--------+-------------------------------------------------+----------------+
```

Figure 17. Reverse proxy receives client request and forwards to localhost:5573

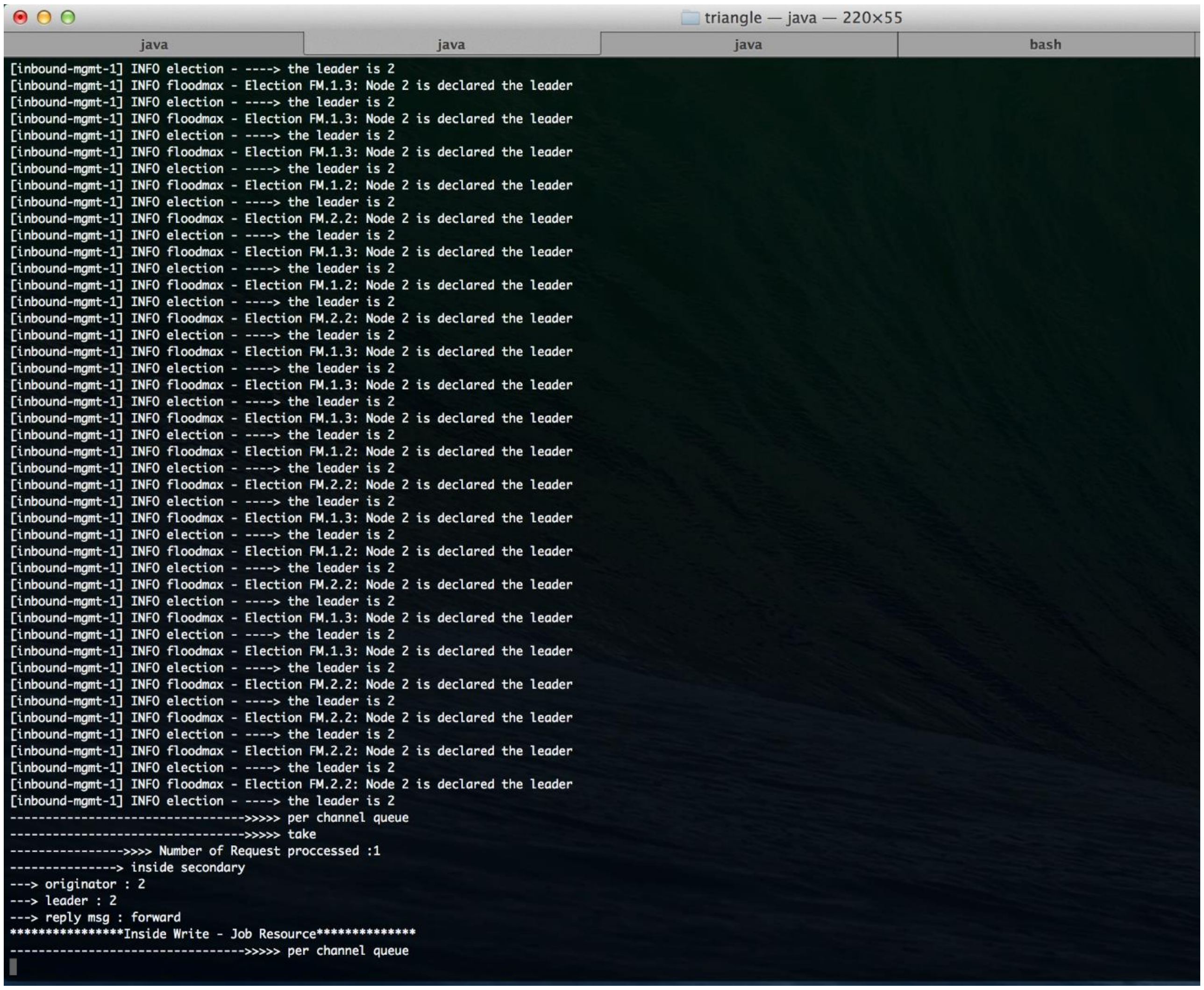
Test Case 3: Receiving Request from Queue and Forwarding to Node



```
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
--------------------------------->>>>> per channel queue
--------------------------------->>>>> take
---------------->>>> Number of Request proccessed :1
roundrobin -- set node0
--------------------------------->>>>> forward resource
--------------------------------->>>>> per channel queue
[nioEventLoopGroup-8-1] INFO server - server is shutting down
[nioEventLoopGroup-8-1] INFO server - server is shutting down
[outbound-1] INFO server - connection queue closing
```

Figure 18. Forward request to node based on Round Robin

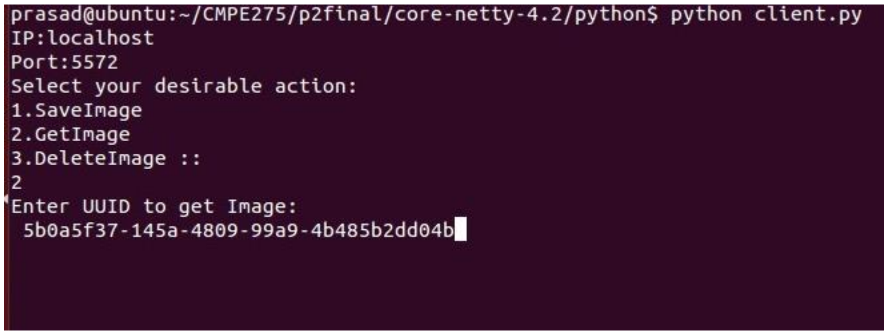
Test Case 4: Image Write After Request is Forwarded



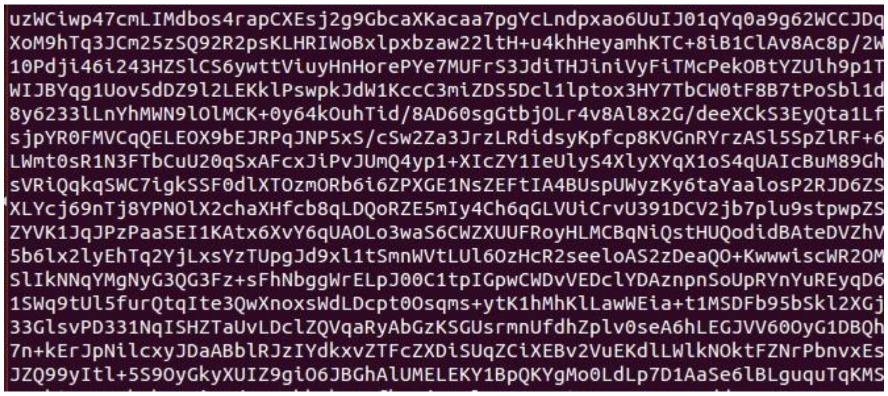
```
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.3: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.3: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.3: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.2.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.3: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.2.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.3: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.3: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.3: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.2.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.3: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.2.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.3: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.1.3: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.2.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.2.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.2.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Election FM.2.2: Node 2 is declared the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
--------------------------------->>>>> per channel queue
--------------------------------->>>>> take
---------------->>>> Number of Request proccessed :1
---------------> inside secondary
---> originator : 2
---> leader : 2
---> reply msg : forward
****************Inside Write - Job Resource**************
--------------------------------->>>>> per channel queue
```

Figure 19. After master node (originator) receives message from client, the master forwards request to next node in round robin for processing

Test Case 5: Read Request and Response



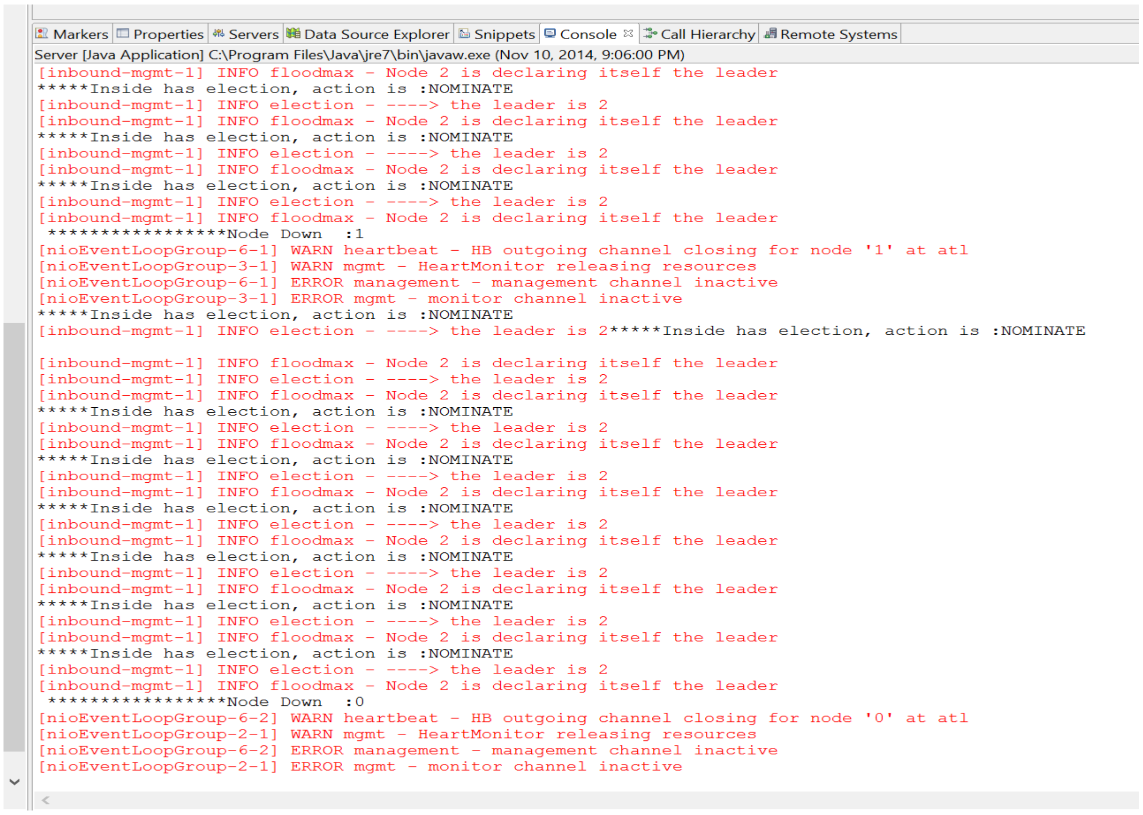
```
prasad@ubuntu:~/CMPE275/p2final/core-netty-4.2/python$ python client.py
IP:localhost
Port:5572
Select your desirable action:
1.SaveImage
2.GetImage
3.DeleteImage ::
2
Enter UUID to get Image:
 5b0a5f37-145a-4809-99a9-4b485b2dd04b
```



```
uzWCiwp47cmLIMdbos4rapCXEsj2g9GbcaXKacaa7pgYcLndpxao6UuIJ01qYq0a9g62WCCJDq
XoM9hTq3JCm25zSQ92R2psKLHRIWoBxlpxbzaw22ltH+u4khHeyamhKTC+8iB1ClAv8Ac8p/2W
10Pdji46i243HZSlCS6ywttViuyHnHorePYe7MUFrS3JdiTHJiniVyFiTMcPekOBtYZUlh9p1T
WIJBYqg1Uov5dDZ9l2LEKklPswpkJdW1KccC3miZDS5Dcl1lptox3HY7TbCW0tF8B7tPoSbl1d
8y6233lLnYhMWN9lOlMCK+0y64kOuhTid/8AD60sgGtbjOLr4v8Al8x2G/deeXCkS3EyQta1Lf
sjpYR0FMVCqQELEOX9bEJRPqJNP5xS/cSw2Za3JrzLRdidsyKpfcp8KVGnRYrzASl5SpZlRF+6
LWmt0sR1N3FTbCuU20qSxAFcxJiPvJUmQ4yp1+XIcZY1IeUlyS4XlyXYqX1oS4qUAIcBuM89Gh
sVRiQqkqSWC7igkSSF0dlXTOzmORb6i6ZPXGE1NsZEFtIA4BUspUWyzKy6taYaalosP2RJD6ZS
XLYcj69nTj8YPNOlX2chaXHfcb8qLDQoRZE5mIy4Ch6qGLVUiCrvU391DCV2jb7plu9stpwpZS
ZYVK1JqJPzPaaSEI1KAtx6XvY6qUAOLo3waS6CWZXUUFRoyHLMCBqNiQstHUQodidBAteDVZhV
5b6lx2lyEhTq2YjLxsYzTUpgJd9xl1tSmnWVtLUl6OzHcR2seeloAS2zDeaQO+KwwwiscWR2OM
SlIkNNqYMgNyG3QG3Fz+sFhNbggWrELpJ00C1tpIGpwCWDvVEDclYDAznpnSoUpRYnYuREyqD6
1SWq9tUl5furQtqIte3QwXnoxsWdLDcpt0Osqms+ytK1hMhKlLawWEia+t1MSDFb95bSkl2XGj
33GlsvPD331NqISHZTaUvLDclZQVqaRyAbGzKSGUsrmnUfdhZplv0seA6hLEGJVV60OyG1DBQh
7n+kErJpNilcxyJDaABblRJzIYdkxvZTFcZXDiSUqZCiXEBv2VuEKdlLWlkNOktFZNrPbnvxEs
JZQ99yItl+5S9OyGkyXUIZ9giO6JBGhAlUMELEKY1BpQKYgMo0LdLp7D1AaSe6lBLguquTqKMS
```

Figure 20. Client Interface for the binary data received during retrieve operation

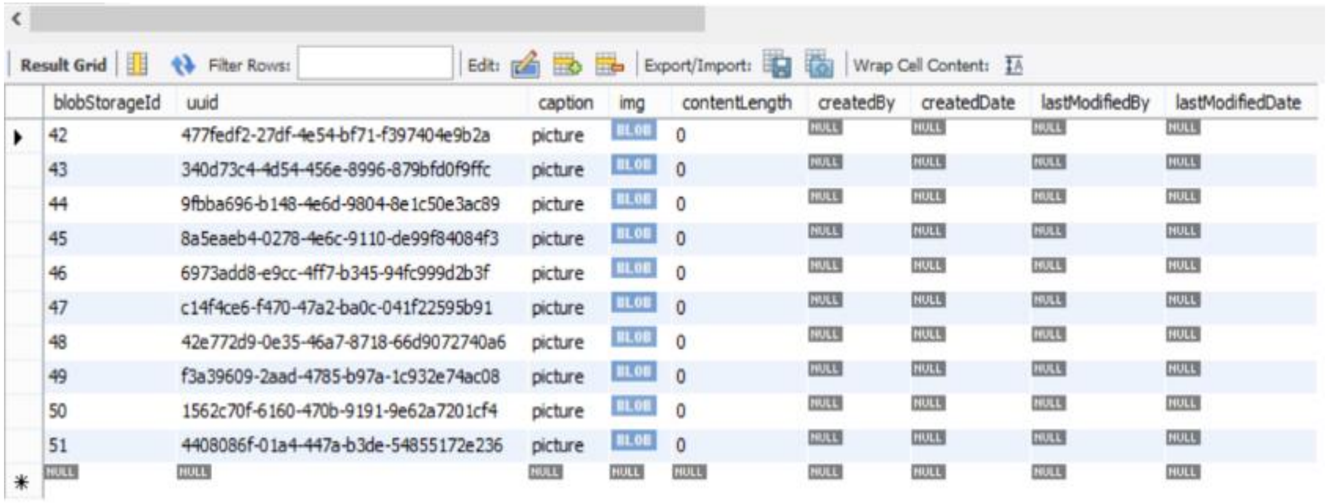
Test Case 6: Re-election After Node Failure



```
Server [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 10, 2014, 9:06:00 PM)
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
 *****************Node Down  :1
[nioEventLoopGroup-6-1] WARN heartbeat - HB outgoing channel closing for node '1' at atl
[nioEventLoopGroup-3-1] WARN mgmt - HeartMonitor releasing resources
[nioEventLoopGroup-6-1] ERROR management - management channel inactive
[nioEventLoopGroup-3-1] ERROR mgmt - monitor channel inactive
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2*****Inside has election, action is :NOMINATE

[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
*****Inside has election, action is :NOMINATE
[inbound-mgmt-1] INFO election - ----> the leader is 2
[inbound-mgmt-1] INFO floodmax - Node 2 is declaring itself the leader
 *****************Node Down  :0
[nioEventLoopGroup-6-2] WARN heartbeat - HB outgoing channel closing for node '0' at atl
[nioEventLoopGroup-2-1] WARN mgmt - HeartMonitor releasing resources
[nioEventLoopGroup-6-2] ERROR management - management channel inactive
[nioEventLoopGroup-2-1] ERROR mgmt - monitor channel inactive
```

Figure 21. Automatic Re-election After Node Failure

Screenshot 1: MySQL Database with Stored Blobs

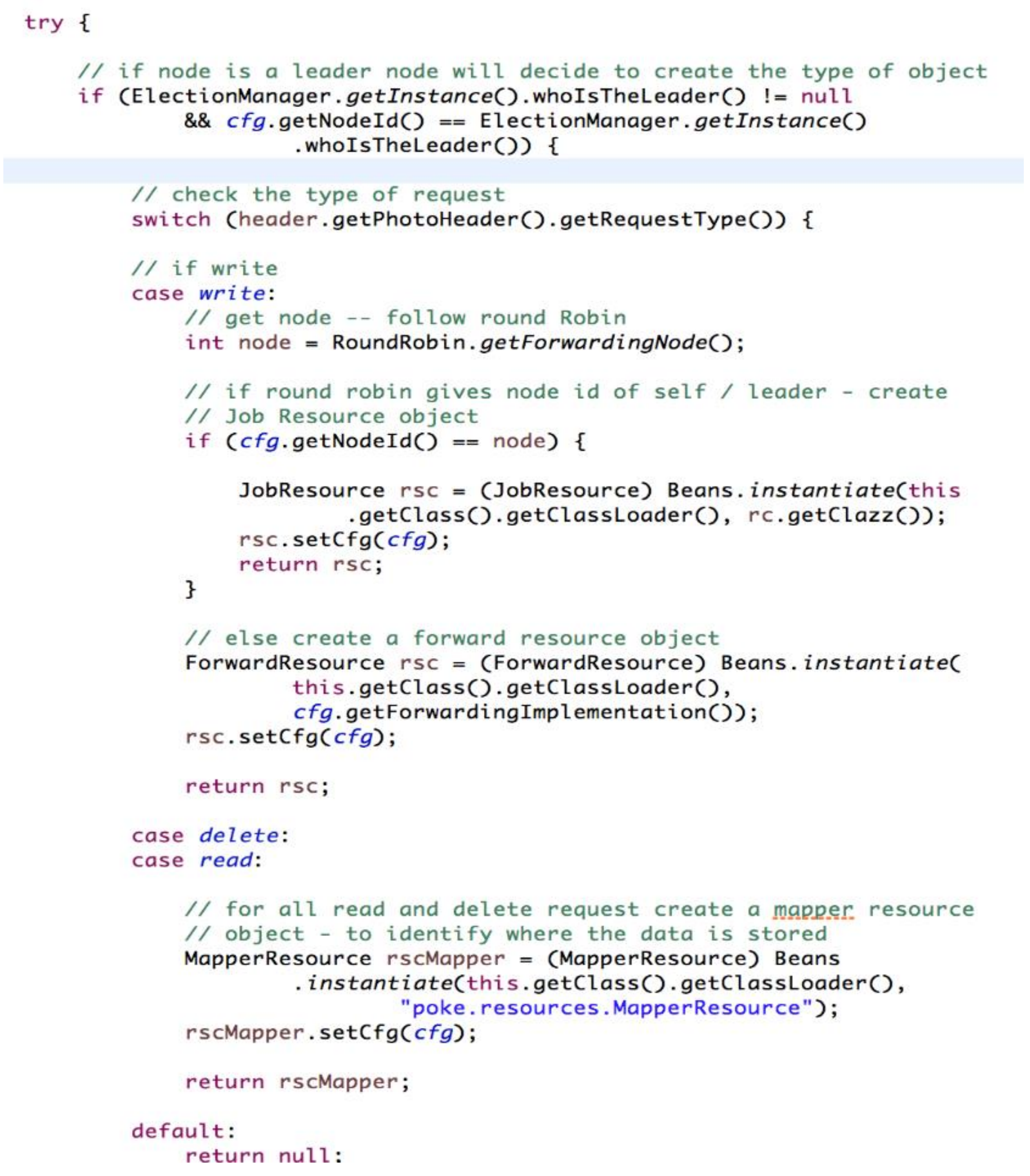


blobStorageId	uuid	caption	img	contentLength	createdBy	createdDate	lastModifiedBy	lastModifiedDate
42	477fedf2-27df-4e54-bf71-f397404e9b2a	picture	BLOB	0	NULL	NULL	NULL	NULL
43	340d73c4-4d54-456e-8996-879bfd0f9ffc	picture	BLOB	0	NULL	NULL	NULL	NULL
44	9fbba696-b148-4e6d-9804-8e1c50e3ac89	picture	BLOB	0	NULL	NULL	NULL	NULL
45	8a5eaeb4-0278-4e6c-9110-de99f84084f3	picture	BLOB	0	NULL	NULL	NULL	NULL
46	6973add8-e9cc-4ff7-b345-94fc999d2b3f	picture	BLOB	0	NULL	NULL	NULL	NULL
47	c14f4ce6-f470-47a2-ba0c-041f22595b91	picture	BLOB	0	NULL	NULL	NULL	NULL
48	42e772d9-0e35-46a7-8718-66d9072740a6	picture	BLOB	0	NULL	NULL	NULL	NULL
49	f3a39609-2aad-4785-b97a-1c932e74ac08	picture	BLOB	0	NULL	NULL	NULL	NULL
50	1562c70f-6160-470b-9191-9e62a7201cf4	picture	BLOB	0	NULL	NULL	NULL	NULL
51	4408086f-01a4-447a-b3de-54855172e236	picture	BLOB	0	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 22. Database with Stored Blobs

Code Snippets

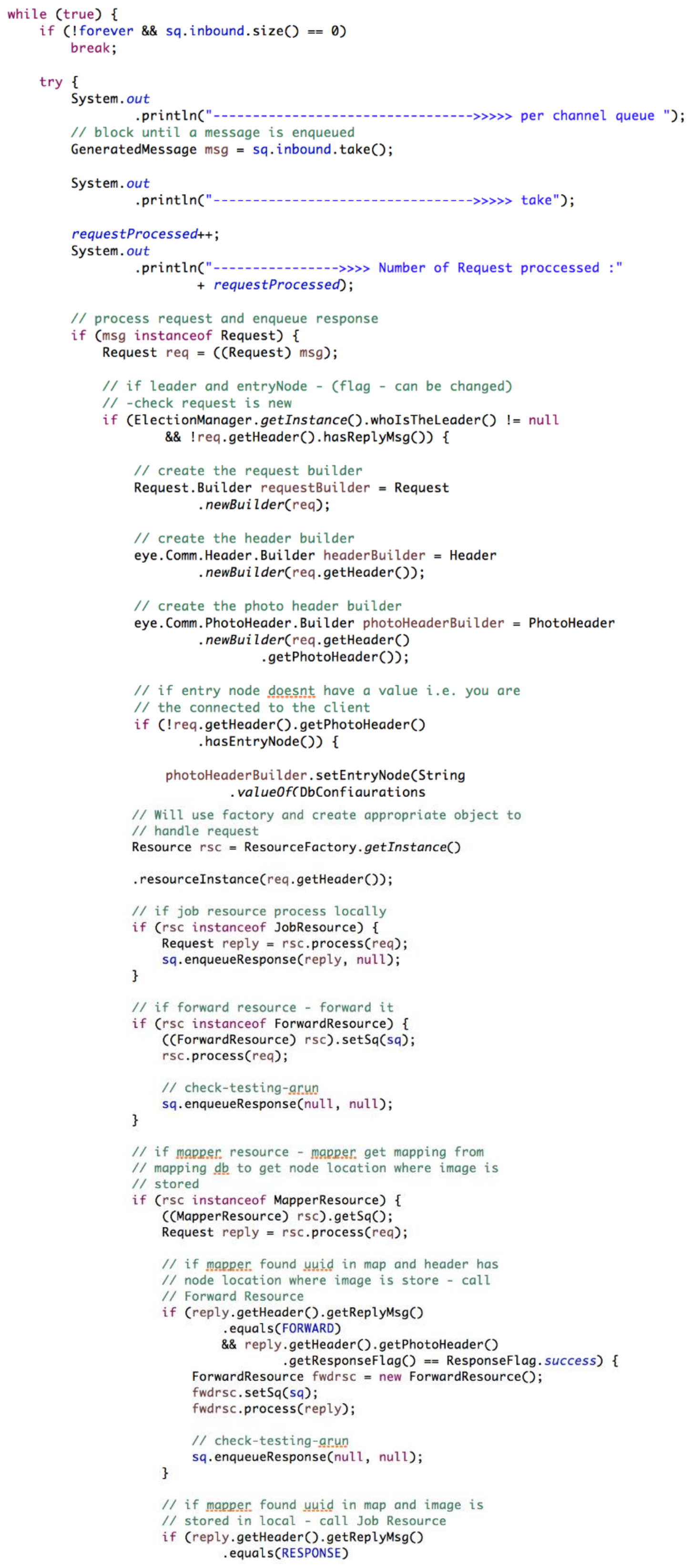
Server Code Snippet: Round Robin Implementation



```
try {

    // if node is a leader node will decide to create the type of object
    if (ElectionManager.getInstance().whoIsTheLeader() != null
            && cfg.getNodeId() == ElectionManager.getInstance()
                    .whoIsTheLeader()) {

        // check the type of request
        switch (header.getPhotoHeader().getRequestType()) {

        // if write
        case write:
            // get node -- follow round Robin
            int node = RoundRobin.getForwardingNode();

            // if round robin gives node id of self / leader - create
            // Job Resource object
            if (cfg.getNodeId() == node) {

                JobResource rsc = (JobResource) Beans.instantiate(this
                        .getClass().getClassLoader(), rc.getClazz());
                rsc.setCfg(cfg);
                return rsc;
            }

            // else create a forward resource object
            ForwardResource rsc = (ForwardResource) Beans.instantiate(
                    this.getClass().getClassLoader(),
                    cfg.getForwardingImplementation());
            rsc.setCfg(cfg);

            return rsc;

        case delete:
        case read:

            // for all read and delete request create a mapper resource
            // object - to identify where the data is stored
            MapperResource rscMapper = (MapperResource) Beans
                    .instantiate(this.getClass().getClassLoader(),
                            "poke.resources.MapperResource");
            rscMapper.setCfg(cfg);

            return rscMapper;

        default:
            return null;
```

Code Snippet 3. Round Robin Implementation

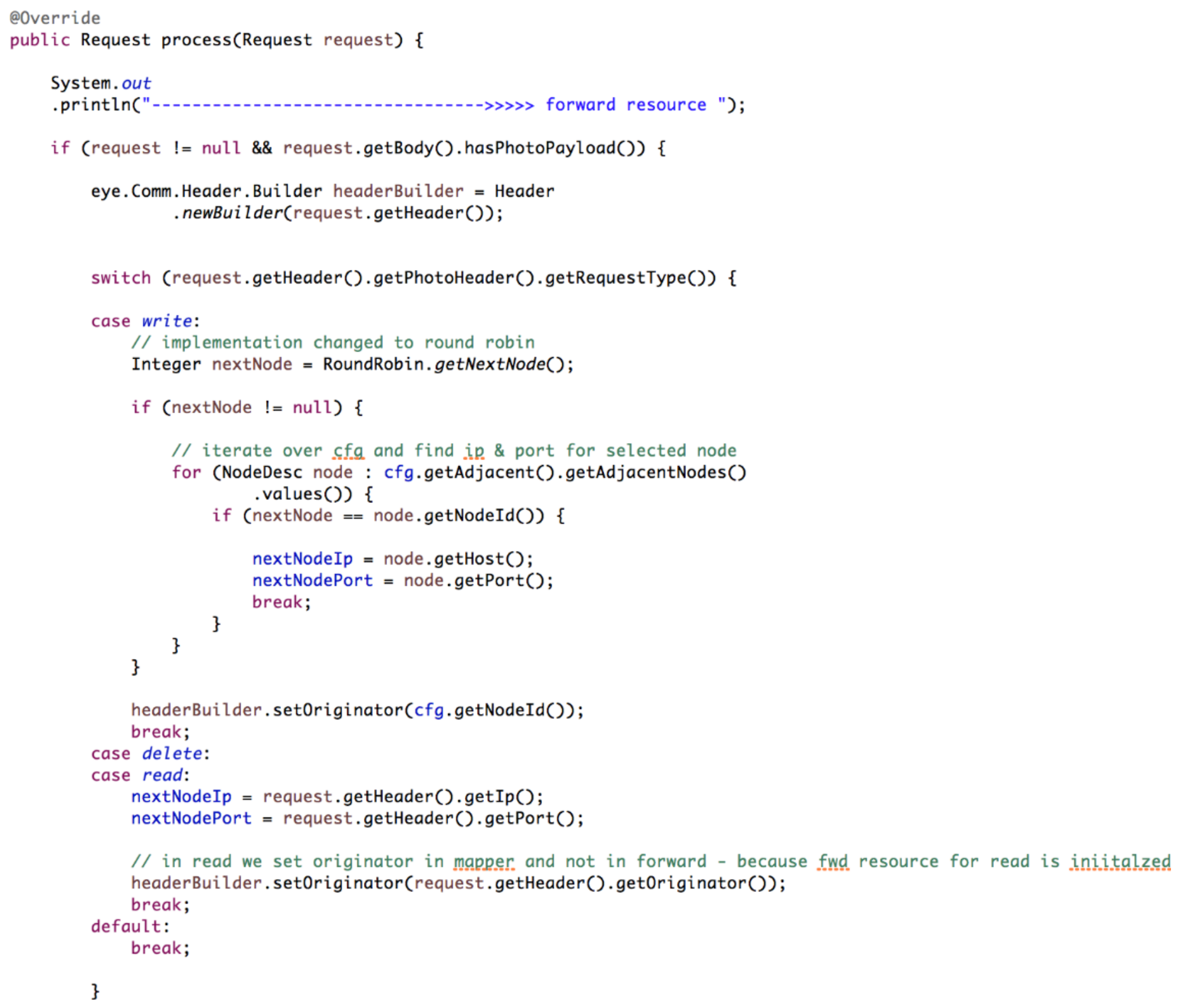
Server Code Snippet: Enqueue Message



```
while (true) {
    if (!forever && sq.inbound.size() == 0)
        break;

    try {
        System.out
                .println("--------------------------------->>>>> per channel queue ");
        // block until a message is enqueued
        GeneratedMessage msg = sq.inbound.take();

        System.out
                .println("--------------------------------->>>>> take");

        requestProcessed++;
        System.out
                .println("--------------->>>> Number of Request proccessed :"
                        + requestProcessed);

        // process request and enqueue response
        if (msg instanceof Request) {
            Request req = ((Request) msg);

            // if leader and entryNode - (flag - can be changed)
            // -check request is new
            if (ElectionManager.getInstance().whoIsTheLeader() != null
                    && !req.getHeader().hasReplyMsg()) {

                // create the request builder
                Request.Builder requestBuilder = Request
                        .newBuilder(req);

                // create the header builder
                eye.Comm.Header.Builder headerBuilder = Header
                        .newBuilder(req.getHeader());

                // create the photo header builder
                eye.Comm.PhotoHeader.Builder photoHeaderBuilder = PhotoHeader
                        .newBuilder(req.getHeader()
                                .getPhotoHeader());

                // if entry node doesnt have a value i.e. you are
                // the connected to the client
                if (!req.getHeader().getPhotoHeader()
                        .hasEntryNode()) {

                    photoHeaderBuilder.setEntryNode(String
                            .valueOf(DbConfiaurations
                // Will use factory and create appropriate object to
                // handle request
                Resource rsc = ResourceFactory.getInstance()

                .resourceInstance(req.getHeader());

                // if job resource process locally
                if (rsc instanceof JobResource) {
                    Request reply = rsc.process(req);
                    sq.enqueueResponse(reply, null);
                }

                // if forward resource - forward it
                if (rsc instanceof ForwardResource) {
                    ((ForwardResource) rsc).setSq(sq);
                    rsc.process(req);

                    // check-testing-grun
                    sq.enqueueResponse(null, null);
                }

                // if mapper resource - mapper get mapping from
                // mapping db to get node location where image is
                // stored
                if (rsc instanceof MapperResource) {
                    ((MapperResource) rsc).getSq();
                    Request reply = rsc.process(req);

                    // if mapper found uuid in map and header has
                    // node location where image is store - call
                    // Forward Resource
                    if (reply.getHeader().getReplyMsg()
                            .equals(FORWARD)
                            && reply.getHeader().getPhotoHeader()
                                    .getResponseFlag() == ResponseFlag.success) {
                        ForwardResource fwdrsc = new ForwardResource();
                        fwdrsc.setSq(sq);
                        fwdrsc.process(reply);

                        // check-testing-grun
                        sq.enqueueResponse(null, null);
                    }

                    // if mapper found uuid in map and image is
                    // stored in local - call Job Resource
                    if (reply.getHeader().getReplyMsg()
                            .equals(RESPONSE)
```

Code Snippet 4. Queue and Message Builder

Server Code Snippet: Request Forwarding



```
@Override
public Request process(Request request) {

    System.out
    .println("-------------------------------->>>>> forward resource ");

    if (request != null && request.getBody().hasPhotoPayload()) {

        eye.Comm.Header.Builder headerBuilder = Header
                .newBuilder(request.getHeader());


        switch (request.getHeader().getPhotoHeader().getRequestType()) {

        case write:
            // implementation changed to round robin
            Integer nextNode = RoundRobin.getNextNode();

            if (nextNode != null) {

                // iterate over cfg and find ip & port for selected node
                for (NodeDesc node : cfg.getAdjacent().getAdjacentNodes()
                        .values()) {
                    if (nextNode == node.getNodeId()) {

                        nextNodeIp = node.getHost();
                        nextNodePort = node.getPort();
                        break;
                    }
                }
            }

            headerBuilder.setOriginator(cfg.getNodeId());
            break;
        case delete:
        case read:
            nextNodeIp = request.getHeader().getIp();
            nextNodePort = request.getHeader().getPort();

            // in read we set originator in mapper and not in forward - because fwd resource for read is iniitalzed
            headerBuilder.setOriginator(request.getHeader().getOriginator());
            break;
        default:
            break;

        }
```

Code Snippet 5. Overwriting the 'process' method for request forwarding

Server Code Snippet: Replicated Image Write

```
ReplicatedDbServiceImplementation mapperService = ReplicatedDbServiceImplementation.getInstance();
try {

    if (request != null && request.getBody().hasPhotoPayload()) {

        switch (request.getHeader().getPhotoHeader().getRequestType()) {
        case write:
            System.out
                    .println("****************Inside Write - Job Resource**************");
            final String uuid = java.util.UUID.randomUUID().toString();

            blob.setCaption(request.getBody().getPhotoPayload()
                    .getName());
            blob.setContentLength(request.getHeader().getPhotoHeader()
                    .getContentLength());
            blob.setImageData(request.getBody().getPhotoPayload()
                    .getData().toByteArray());
            blob.setUuid(uuid);
            mapper.setUuid(uuid);
            mapper.setNodeId(cfg.getNodeId());

            try {

                // create blob in db
                BlobStorageServiceImplementation.getInstance()
                        .createBlobStorage(blob);

                // create blob in mapper db
                mapperService.createMapperStorage(mapper);

                // create a body - photo payload builder from request
                Builder photoPayLoadBuilder = PhotoPayload
                        .newBuilder(request.getBody().getPhotoPayload());

                // create a body - payload builder from request
                eye.Comm.Payload.Builder payLoadBuilder = Payload
                        .newBuilder(request.getBody());

                // create the request builder
                Request.Builder requestBuilder = Request
                        .newBuilder(request);

                // create the photo header builder
                eye.Comm.PhotoHeader.Builder photoHeaderBuilder = PhotoHeader
                        .newBuilder(request.getHeader()
                                .getPhotoHeader());
```

Code Snippet 6. Replicated image write job

Server Code Snippet: Blob Storage Profile

```
public class BlobStorageProfile {
        private Long blobStorageId;
        private String uuid;
        private String caption;
        private byte[] imageData;
        private Integer contentLength;
        private String createdBy;
        private Date createdDate;
        private String lastModifiedBy;
        private Date lastModifiedDate;
        public BlobStorageProfile()
           {}

        public BlobStorageProfile(BlobStorage blob)
           {

              this.blobStorageId = blob.getBlobStorageId();
              this.caption = blob.getCaption();
              this.contentLength = blob.getContentLength();
              this.createdBy = blob.getCreatedBy();
              this.createdDate = blob.getCreatedDate();
              this.imageData = blob.getImageData();
              this.lastModifiedBy=blob.getLastModifiedBy();
              this.lastModifiedDate = blob.getLastModifiedDate();
              this.uuid = blob.getUuid();

           }
```

Code Snippet 7 Blob Storage Structure for Correlating Information Received in Protobuf File Format

Server Code Snippet: Insert Blob into DB

```
public BlobStorageProfile createBlobStorage(BlobStorage blob)
                  throws Exception {

            conn = getConnection();

            stmt = conn.createStatement();
            PreparedStatement ps = null;

            try {

                  conn.setAutoCommit(false);
                  String sql = "INSERT INTO blobStorage
(`uuid`,`caption`,`img`,`contentLength`,`createdBy`,`createdDate`
,`lastModifiedBy`,`lastModifiedDate`) VALUES(?,?,?,?,?,?,?,?)";

                  ps = conn.prepareStatement(sql);
                  ps.setString(1, blob.getUuid());
                  ps.setString(2, blob.getCaption());
                  ps.setBytes(3, blob.getImageData());
                  ps.setLong(4, blob.getContentLength());
                  ps.setString(5, blob.getCreatedBy());
                  ps.setDate(6, (Date) blob.getCreatedDate());
                  ps.setString(7, blob.getLastModifiedBy());
                  ps.setDate(8, (Date)
blob.getLastModifiedDate());

                  ps.executeUpdate();
                  conn.commit();

            } finally {
                  ps.close();
                  conn = null;
            }

            return findByUuid(blob.getUuid());
        }
```

Code Snippet 8. Insert Blob into DB After Appropriate Parsing Input

Server Code Snippet: Delete Blob

```
    public Boolean deleteBlobStorage(Long blobStorageId) throws
Exception {

            PreparedStatement ps = null;
            Boolean success = false;
            conn = getConnection();

            try {

                String sql = "DELETE FROM blobStorage where
blobStorage.blobStorageId = ?;";

                ps = conn.prepareStatement(sql);
                ps.setLong(1, blobStorageId);

                ps.executeQuery();
                success = true;
                return success;

            } catch (Exception ex) {
                success = false;
                return success;
            } finally {
                ps.close();
                conn = null;
            }
        }
```

Code Snippet 9. Delete Blob into DB Based on ID

Server Code Snippet: Find Blob by UUID

```
    public BlobStorageProfile findByUuid(String uuid) throws
Exception {

            PreparedStatement ps = null;
            BlobStorage blob = null;
            conn = getConnection();

            try {

                String sqlSelect = "SELECT * FROM blobStorage
where blobStorage.uuid = ?;";

                ps = conn.prepareStatement(sqlSelect);
                ps.setString(1, uuid);

                ResultSet rs = ps.executeQuery();
```

```
                    if (rs.next()) {
                          blob = new BlobStorage();

            blob.setBlobStorageId(rs.getLong("blobStorageId"));
                          blob.setUuid(rs.getString("uuid"));
                          blob.setCaption(rs.getString("caption"));

            blob.setContentLength(rs.getInt("contentLength"));
                          blob.setImageData(rs.getBytes("img"));

            blob.setCreatedBy(rs.getString("createdBy"));

            blob.setCreatedBy(rs.getString("lastModifiedBy"));

            blob.setCreatedDate(rs.getDate("createdDate"));

            blob.setLastModifiedDate(rs.getDate("lastModifiedDate"));
                    }

                    return new BlobStorageProfile(blob);
              } finally {
                    ps.close();
                    blob = null;
                    conn = null;
              }
      }
```

Code Snippet 10. Find Blob Based on UUID and Return DB Results

(Note: There are other search options, such as by 'createdBy' field and so forth)

Server Code Snippet: Delete Blob by UUID

```
            @Override
            public Boolean deleteBlobStorageByUuid(String uuid) throws
      Exception {
                  PreparedStatement ps = null;
                  Boolean success = false;
                  conn = getConnection();

                  try {

                        String safeSql = "SET SQL_SAFE_UPDATES=0;";
                        ps = conn.prepareStatement(safeSql);
                        ps.executeQuery();


                        String sql = "DELETE FROM blobStorage where
      blobStorage.uuid = ?";

                        ps = conn.prepareStatement(sql);
                        ps.setString(1, uuid);

                        ps.executeQuery();

                        success = true;
                        return success;
                  } catch (Exception ex) {
                        success = false;
                        return success;
                  } finally {
                        ps.close();
                        conn = null;

                  }

            }
```

Code Snippet 11. Delete Blob Based on UUID by Updating DB and Setting Boolean

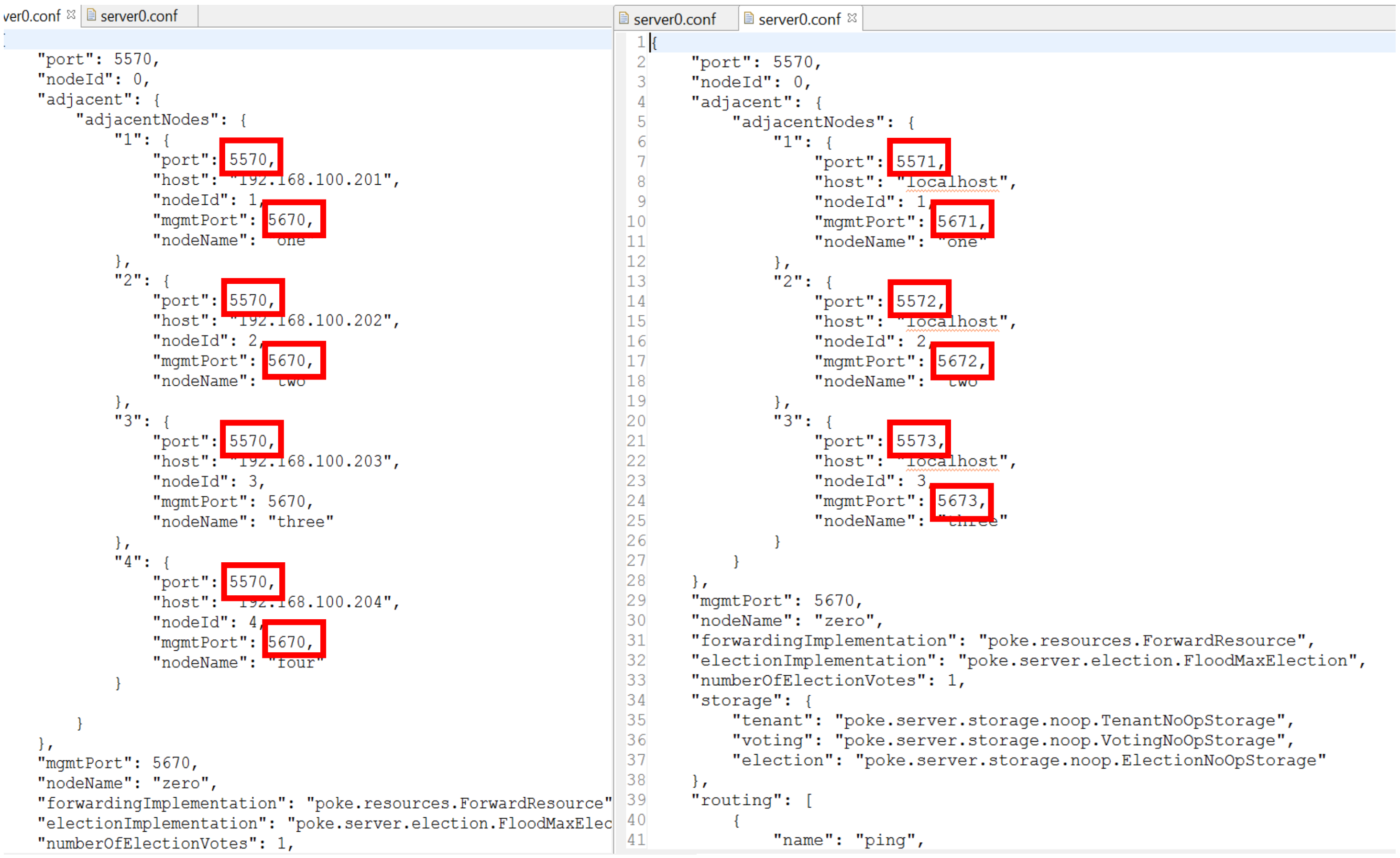
(Note: There are other delete options, such as by 'createdBy' field and so forth)

Server Code Snippet: Update Cluster Mapper

```
public Boolean updateClusterMapper(int clusterId, String host,
int port) throws Exception {

            PreparedStatement ps = null;
            Boolean success = false;
            conn = getDbConnection();

            try {

                  String sql = "UPDATE cmpe275.clusterMapper cm
SET cm.leaderHostAddress = ?, cm.port = ? where cm.clusterId =
?";
                  ps = conn.prepareStatement(sql);
                  ps.setString(1, host);
                  ps.setInt(2, port);
                  ps.setInt(3, clusterId);

                  ps.executeQuery();
                  success = true;
                  return success;

            } catch (Exception ex) {
                  success = false;
                  return success;
            } finally {
                  ps.close();
                  conn = null;
            }
      }
```

Code Snippet 12. Shared Replicated Database to Correlate UUID to Corresponding Node

Server Code Snippet: Database Configuration Files for Mesh Topology



```
ver0.conf   server0.conf

    "port": 5570,
    "nodeId": 0,
    "adjacent": {
        "adjacentNodes": {
            "1": {
                "port": 5570,
                "host": "192.168.100.201",
                "nodeId": 1,
                "mgmtPort": 5670,
                "nodeName": "one"
            },
            "2": {
                "port": 5570,
                "host": "192.168.100.202",
                "nodeId": 2,
                "mgmtPort": 5670,
                "nodeName": "two"
            },
            "3": {
                "port": 5570,
                "host": "192.168.100.203",
                "nodeId": 3,
                "mgmtPort": 5670,
                "nodeName": "three"
            },
            "4": {
                "port": 5570,
                "host": "192.168.100.204",
                "nodeId": 4,
                "mgmtPort": 5670,
                "nodeName": "four"
            }

        }
    },
    "mgmtPort": 5670,
    "nodeName": "zero",
    "forwardingImplementation": "poke.resources.ForwardResource"
    "electionImplementation": "poke.server.election.FloodMaxElec
    "numberOfElectionVotes": 1,
```

```
server0.conf   server0.conf

{
    "port": 5570,
    "nodeId": 0,
    "adjacent": {
        "adjacentNodes": {
            "1": {
                "port": 5571,
                "host": "localhost",
                "nodeId": 1,
                "mgmtPort": 5671,
                "nodeName": "one"
            },
            "2": {
                "port": 5572,
                "host": "localhost",
                "nodeId": 2,
                "mgmtPort": 5672,
                "nodeName": "two"
            },
            "3": {
                "port": 5573,
                "host": "localhost",
                "nodeId": 3,
                "mgmtPort": 5673,
                "nodeName": "three"
            }
        }
    },
    "mgmtPort": 5670,
    "nodeName": "zero",
    "forwardingImplementation": "poke.resources.ForwardResource",
    "electionImplementation": "poke.server.election.FloodMaxElection",
    "numberOfElectionVotes": 1,
    "storage": {
        "tenant": "poke.server.storage.noop.TenantNoOpStorage",
        "voting": "poke.server.storage.noop.VotingNoOpStorage",
        "election": "poke.server.storage.noop.ElectionNoOpStorage"
    },
    "routing": [
        {
            "name": "ping",
```

Code Snippet 13. Comparison to Show P2P Design and Difference from Initial DB Setup

Client Code Snippet: Save Image

```
def buildSaveImageJob(iname, data, ownerId):

    jobId = str(int(round(time.time() * 1000)))
    r= comm_pb2.Request()

    r.header.photoHeader.requestType = 1    #1 is for write
    r.body.photoPayload.name=iname
    r.body.photoPayload.data = data

    r.header.originator = 0
    r.header.routing_id = comm_pb2.Header.JOBS
    r.header.toNode = 1
    msg = r.SerializeToString()
    return msg
```

Code Snippet 14. Save Image and Establish Corresponding 'jobId'

Client Code Snippet: Retrieve Image

```
def buildGetImageJob(gImage):
     r= comm_pb2.Request()

     r.header.photoHeader.requestType = 0    #0 is the requestType for read
     r.body.photoPayload.uuid=gImage

     r.header.originator = 0
     r.header.routing_id = comm_pb2.Header.JOBS
     r.header.toNode = 1
     msg=r.SerializeToString()
     return msg
```

Code Snippet 15. Retrieve Image by Finding Corresponding UUID for Given Image Name

Client Code Snippet: Delete Image

```
def buildDeleteImageJob(dImage):
     r= comm_pb2.Request()
     r.header.photoHeader.requestType = 2  #2 is the requestType for  delete
     r.body.photoPayload.uuid=dImage

     r.header.originator = 0
     r.header.routing_id = comm_pb2.Header.JOBS
     r.header.toNode = 1
     msg=r.SerializeToString()
     return msg
```

Code Snippet 16. Delete Image by Finding Corresponding UUID for Given Image Name

Client Code Snippet: Send Message

```
def sendMsg(msg_out, port, host):
    s = socket.socket()
#    host = socket.gethostname()
#    host = "192.168.0.87"

    s.connect((host, port))
    msg_len = struct.pack('>L', len(msg_out))
    s.sendall(msg_len + msg_out)
    len_buf = receiveMsg(s, 4)
    msg_in_len = struct.unpack('>L', len_buf)[0]
    msg_in = receiveMsg(s, msg_in_len)

    r = comm_pb2.Request()
    r.ParseFromString(msg_in)
#    print msg_in
#    print r.body.job_status
#    print r.header.reply_msg
#    print r.body.job_op.data.options
    s.close
    return r
```

Code Snippet 17. Structure Message Packet to be Sent to Server and Display Status

Client Code Snippet: Receive Messages and Get Job Type

```
        def receiveMsg(socket, n):
            buf = ''
            while n > 0:
                data = socket.recv(n)
                if data == '':
                    raise RuntimeError('data not received!')
                buf += data
                n -= len(data)
            return buf

        if __name__ == '__main__':
            host = raw_input("IP:")
            port = raw_input("Port:")

            #port = 5573 #raw_input("Port:")
            port = int(port)
            whoAmI = 1;

            input = raw_input("Select your desirable
        action:\n1.SaveImage\n2.GetImage\n3.DeleteImage ::\n")

             if input == "1":
                 fh = open('picture.JPG','rb')
                 dataFile = fh.read()
                 iname="picture"
    fh.close()
     data = base64.b64encode(dataFile)    #Encoding the read binary file
     saveimageJob = buildSaveImageJob(iname, data, 1)
     result = sendMsg(saveimageJob, port, host)
     print("The image has been saved successfully !!::\n")
     print("UUID generated for the saved image::\n")
     print result.body.photoPayload.uuid


elif input == "2":
     gImage = raw_input("Enter UUID to get Image:\n ")
     getImagejob=buildGetImageJob(gImage)
     result = sendMsg(getImagejob, port, host)
     print result.body.photoPayload.uuid
     print("The name of image retrieved ::\n")        #additional part
     print result.body.photoPayload.name
     print("The retrieved image::\n")
     print result.body.photoPayload.data


elif input == "3":
```

```
dImage= raw_input("Enter UUID for image to delete:\n ")
delImagejob=buildDeleteImageJob(dImage)
result=sendMsg(delImagejob, port, host)
print("The response flag received:\n")
print result.header.photoHeader.responseFlag
if result.header.photoHeader.responseFlag == 0:#0 represents success
    print("The image has been deleted successfully !!")
elif result.header.photoHeader.responseFlag == 1:
    print("The image was not deleted successfully !!")
```

Code Snippet 18. Retrieve Message from Port and Get Job Type from Client

Client Code Snippet: Retrieve Request from Elected Leader

```
def getBroadcastMsg(port):
    # listen for the broadcast from the leader"

    sock = socket.socket(socket.AF_INET,  # Internet
                         socket.SOCK_DGRAM)  # UDP

    sock.bind(('', port))

    data = sock.recv(1024)  # buffer size is 1024 bytes
    return data
```

Code Snippet 19. Bind Port and Retrieve Message for Worker Node to Complete Request

Client Code Snippet: Create Ping to Establish Connection

```
def buildPing(tag, number):

    r = comm_pb2.Request()

    r.body.ping.tag = str(tag)
    r.body.ping.number = number


    r.header.originator = 0
    r.header.tag = str(tag + number + int(round(time.time() * 1000)))
    r.header.routing_id = comm_pb2.Header.PING
    r.header.toNode = 1

    msg = r.SerializeToString()
    return msg
```

Code Snippet 20. Ping Node to Determine Ability to Process Job

Closing Remarks

One of the main troubles of this project is the communication, specifically the lack thereof. This project requires all of the teams within the class to agree on two things: a common protocol buffer structure and the interface to communicate with other clusters. This problem could be a result of the number of team leaders, “too many cooks in the kitchen” as it were, trying to decide the project requirements. To add to the frustrations, it’s a logistical nightmare trying to coordinate a time and place for a meeting between the fourteen team leads, and few leaders are willing to participate once a meeting is scheduled. One way to address this issue, to enforce a deadline on the above two requirements ahead of time in the hopes that people take a greater interest in the project with a deadline looming. This was already attempted within in the group but it failed as the deadline was a communal deadline and not an assignment for the class. Another resolution is to break the team leaders into smaller team that address a smaller issue and regroup to solve the bigger issue. Besides the communication difficulties, the two technologies (Netty and ProtoBuf) are not as well documented as a developer could of have hoped for. Both technologies offer many sample “getting-started” projects, but they lack the necessary documentation to configure more complex applications, such as this project.

If we were to attempt to do this project again with a proto file and a communication interface already defined, we would like to create the project from scratch. We used existing code and a proto file. The code has many complicated features and exceptions that were unnecessary for our project goal causing us to waste time trying to figure out how the code functioned and how to fix the exceptions. The proto file that also had many attributes that raised many questions but answered few. In the end and to our detriment, we believed that it was simpler to work with the existing code rather than “reinventing the wheel.” Another avenue of research for this project would be to using consistent hashing to distribute the data load. We already know from the Amazon’s Dynamo publication that consistent hashing does not solve a problem in and of itself but can be used if the nodes were in a hybrid sharded-replicated network where multiple nodes can handle a request, not just one.

References

[1] D. Thakur, "What is Data Replication? Advantages & Disadvantages of Data Replication.," E-Computer Notes, 2014. [Online]. Available: http://ecomputernotes.com/database-system/adv-database/data-replication. [Accessed 6 November 2014].

[2] A. Silberschatz, "Distributed Databases," Yale, 2012. [Online]. Available: Powerpoint.

[3] B. Murgante, "Computational Science and Its Applications," in *ICCSA 2014: 14th International Conference*, Guimarães, Portugal, 2014.

[4] CodeFutures, "Database Sharding," 2014. [Online]. Available: http://codefutures.com/database-sharding/. [Accessed 8 November 2014].

[5] J. Wightman, "Optimization for SSOS," in *Pro SQL Server 2005 IntegrationServices*, Springer, New York, 2005.

[6] TechEmpower, "Web Framework Benchmarks," TechEmpower, 01 05 2014. [Online]. Available: http://www.techempower.com/benchmarks/#section=data-r9&hw=i7&test=plaintext. [Accessed 11 11 2014].

[7] N. Maurer, "Why Netty?," 2014.

[8] N. Maurer, "Netty 4 > Intro > Changes > HTTP > Lessons Learned," CA, 2014.

[9] I. Annischenko, "Protobuf, Thrift, and Avro," 2012. [Online]. Available: http://www.slideshare.net/IgorAnishchenko/pb-vs-thrift-vs-avro.

[10] Stack Overflow, *MySQL Sharding Approaches,* 2011.

[11] M. Kamel and e. al., "A Survey of Structured P2P Systems for RDF Data Storage and Retrival," *Transaction on Large-Scale Data and Knowledge Centered Systems III: Special Issue on Data an dKNowledge Management in Grid and PSP Systems,* p. 21, 2011.

[12] M. Castro, M. Costa and A. Rowstron, "Peer-to-Peer Overlays: Structured, Unstructured, or both?," Microsoft Corporation, Redmond, 2004.

[13] Oracle, "Master Replication Concetps and Architecture," [Online]. Available: http://docs.oracle.com/cd/B28359_01/server.111/b28326/repmaster.htm#BGBHCCDJ. [Accessed 2014 November 4].

[14] P. Sawers, "Samsung Ventures continues its investment offensive with DBaaS company Cloudant," TNW, 6 February 2013. [Online]. Available: http://thenextweb.com/insider/2013/02/06/samsung-ventures-adds-dbaas-company-cloudant-to-its-investment-portfolio/. [Accessed 2014 November 3].