IT 11 075
Examensarbete 30 hp
Oktober 2011

Evaluation and Implementation of Distributed NoSQL Database for MMO Gaming Environment

Yousaf Muhammad

Institutionen för informationsteknologi
Department of Information Technology

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Evaluation and Implementation of Distributed NoSQL Database for MMO Gaming Environment

Yousaf Muhammad

Massively Multi-player Online Games have emerged as a most intensive data application nowadays. Being massively used by simultaneously game players around the world. This data require high level of performance, fault tolerance and scalability. Distributed databases are one of the option we got for this kind of systems. The goal is to give game players a high level of availability, fault tolerance and speed of the database. So that the growing need of the players can be handled. NoSQL distributed databases are nowadays are getting popularity for their opensourse, non relational data stores, high performance, scalability and fault tolerance. Traditional relational databases like MySQL, PostgreSQL seems to be loosing interest among the database developer. NoSQL databases which includes Riak, CouchDB, Cassandra are rapidly gaining interest because of there advantages over traditional databases. This study is about choosing the best and reliable distributed database among SQL and NoSQL.

Pikkotekk AB provides network traffic load balancing services to the game developers for their games. PikkoTekk have load balancing servers known as Pikkoservers that works fully transparent to the game and fulfill all the requirement of massively multi player online user that interacts with game all together.

Handledare: Christian Lönnholm
Ämnesgranskare: Justin Pearson
Examinator: Anders Jansson
IT 11 075
Tryckt av: Reprocentralen ITC

Contents

Acknowledgement:

First of all, I would like to praise to almighty ALLAH for all the favours and mercifulness that bestowed upon me not only for completing my Master thesis project but for everything that I have had achieved yet.

Secondly, I want to acknowledge my thanks to my supervisor Christian Lonnholm, Bjorn Dahlman and my reviewer Justin Pearson for helping me out at different stages of my project.

Finally, my gratitude goes to my parents who always help me in achieving my desires in life.

1 Introduction

1.1 Research background

Traditional SQL RDBMS (Structured Query Language Relational Database Management System) are hard to scale to the sheer amount of data and how to connect internally. One way to address this problem is by using NoSQL database. The more interactivity of the gaming environment in modern days have attracted more game users. In effect, the data generated through such multi-player network games are increasing rapidly. So proper data management and data handling is the need of today for improving speed, reliability and scalability of these gaming applications. Distributed database is the best bet among all.

PikkoTekk AB [PKT] provides state of the art network traffic load balance solutions to the video games. So that unlimited number of players can join one game or multi-game at one time.

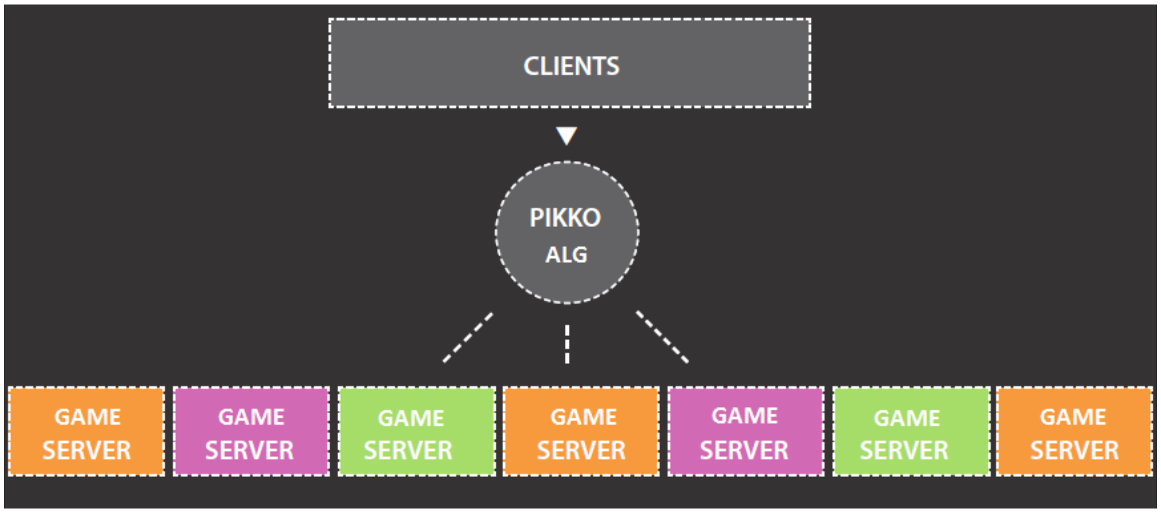

Figure 1: PikkoTekk Architecture

Figure 1 shows the Pikko architecture that has the ability to handle large number of identical single-threaded game server. Pikko architecture provides appealing programming environment for the game developers and game servers developer by providing them an option to have a power of horizontal scalability by running game server on single thread.

Any database whether its a SQL or NoSQL can fit into Pikko architecture. For high performance, scalability and fault tolerance modern distributed NoSQL database is recommended.

Game Server: The Pikko server software consists of Pikko server(Game Sever) and several cell servers. Player in an online multiplayer game connect to the Pikko server, which handles load balancing between cell servers. The cell servers handles physics, game logic and more.

LobbyServer: Lobby server gives user the platform to host the game, switching to other game, chatting among users in network and enjoy other features apart from playing games only. It makes them socialize and have an interactive gaming environment to spend more time on gaming.

1.2 Task of the thesis:

Video games industry has had a rapid growth over the past few years. According to the statistics by [PKT], the sales of video games were $16 billion in 2007 and it is expected to grow over in upcoming years. This is due to fact that technological growth which include high speed broadband take over the slow dial up network access, video games in portable devices, multi player online games and rapid development of games by game developers.

The aim of this thesis is to perform exhaustive comparison of NoSQL databases. Further, evaluate these distributed NoSQL databases by their performance and advantages in distributed environment. Finally, design and implementation of the database by testing it on real time distributed application.

Selection of the Database for the distributed application will depend upon the performance and scalability of the compared NoSQL database.

In MMO Massively Multi player modern online games, apart from playing games socializing through chat, joining and switching between different game environments, automated and manual hosting of games are common features. To keep track of such large amount of user data in between sessions we normally use a database backend. As the user activities normally span multiple servers, a distributed database suits this task well. As scalability, availability and concurrency puts high pressure on the persistence of the database, a smart design is needed to make transactions fluent.

1.3 Outline of Thesis:

1. Chapter 2 defines SQL, NoSQL and key difference between them.
2. Chapter 3 explains theoretical background which includes, Distributed Systems and comparison between some popular SQL and NoSQL databases.
3. Chapter 4 gives a brief introduction about Riak.
4. Chapter 5 introduces the process of Implementation and introduction of tools used.
5. Chapter 6 introduces a tests and performance evaluation of the databases.
6. Chapter 7 concludes the thesis.

2 Theoretical Basis:

This chapter will explains some important concepts that relate to the thesis.

2.1 SQL: (Structured Query Language)

SQL is a one of many database language that can be used for querying and modifying relational databases. In Relational databases data is stored in RDBMS (Relational Database Management System).

> A database management system is a set of software programs that controls the organization, storage, management and retrieval of data in a database. [DBM]
>
> in Wikipedia 25/06/2011

In RDBMS data is stored in database tables. These tables are database objects and the most common form of data storage in RDBMS. Each table in the database is divided in to small entities called field. Fields are also called as columns of the data. Rows of the data consist of each individual entry in fields of the data.

Some RDBMS that uses SQL as there data manipulation language are Microsoft SQL server, Oracle, MySQL, Mircosoft Access and Sybase etc. The most common and widely used SQL statements are Select, Insert, Update, Delete, Create and Drop. By using these common and standard statements we can do pretty much anything with the database.

Microsoft SQL Server: Microsoft SQL Server is one of the most popular database among relation databases. SQL server is not only a database it is also a complete management system. Apart from the common functionality of the database SQL server also comes with the addition tools like manipulation, report writing, data import export, data structure and management.

Oracle: Oracle is considered as world leading relational database. Oracle was the first database that support SQL as a manipulation language. [SOra]

2.2 Difference Between NoSQL and SQL:

NoSQL is an umbrella term designating a group of non-relational database systems or key-value stores. A common trait to these systems is that usually they, in contrast to relational databases do not need a fixed table structure. Their strong point is their distributivity, making them particularly suitable for scaling up. Traditional database systems are known for their continuous consistency, which is traded for better availability or partition tolerance in NoSQL databases.

The key difference between NoSQL and SQL is that NoSQL provide schema less data model that results in faster read and writes in the database as compared to SQL.

Some of the commonly used NoSQL databases are CouchDB, Riak, Cassandra, Mnesia, BerkeleyDB, HamsterDB, MongoDB and Redis etc. All NoSQL databases have there own advantages and drawbacks in term of storage, performance and availability.

[LND] describes families and list of complete set of NoSQL databases.

Mnesia: Mnesia is a NoSQL database that is considered as an Object relational DBMS. It has all the characteristics of DBMS which includes Locking, replication, logging, primary and secondary memory storage etc.

Cassandra: Cassandra is known for its high scalability, decentralized nature, durability, fault tolerance and linear in performance on adding new machine.

3 Analysis

In chapter 3, theoretical background is outlined. This chapter will discuss the design, implementation of the database and interaction with the multiplayer game. In the next chapter comparison between different database are done and also database to be implemented will be selected.

3.1 Requirement of Distributed Database:

A distributed database is a database that has no single physical location of its own. Instead, it is distributed across the network of many computers. These computers are distributed geographically by connecting them through communication links. User of the database thinks it as a centralized system but it is not in reality. The main disadvantage of centralized databases is their vulnerability for downlink. Contrary of that, distributed databases have advantage of fault tolerance. Generally speaking

> Distributed database is a collection of multiple, logically interrelated databases distributed over a computer network. A distributed database management system (distributed DBMS) is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users. [PDDS,Chap.1]

A simple distributed database architecture is shown in figure 2

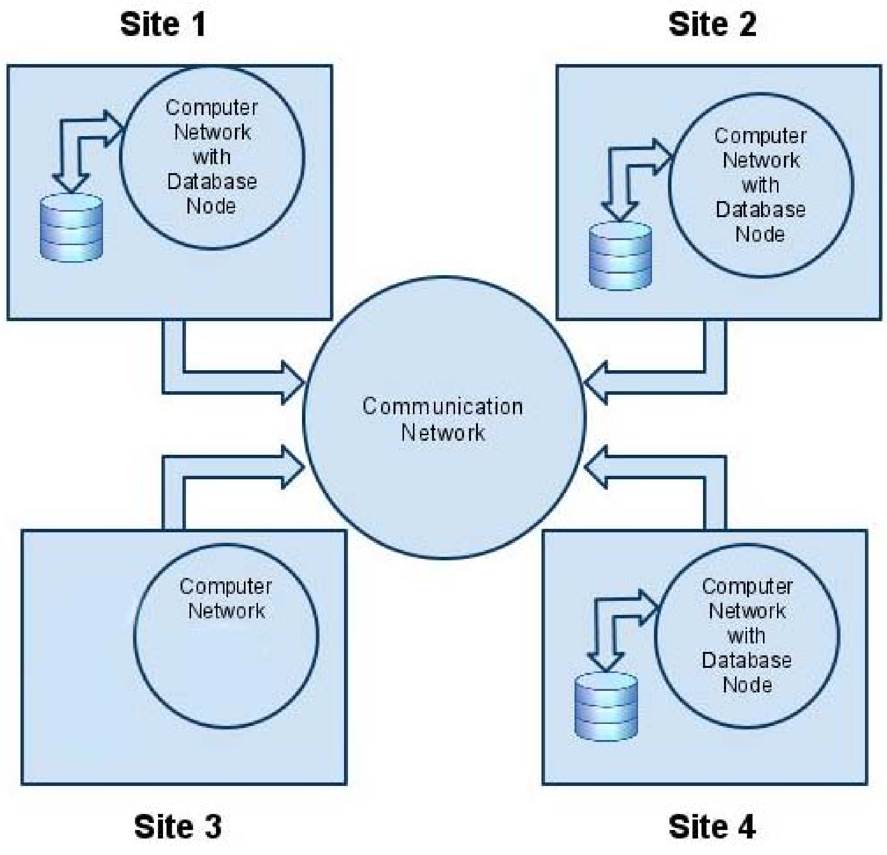
Site 1
Computer Network with Database Node
Site 2
Computer Network with Database Node
Communication Network
Computer Network
Computer Network with Database Node
Site 3
Site 4

Figure 2: Distributed Database

In distributed databases data is shared among all the connected nodes that allows faster local queries and subject to reduce network traffic.

3.1.1 Distributed Database Advantages:

This section will discuss some of the benefits of distributed database systems. According to [PDDS Chap.1] advantages of DDBS are:

Transparent Management of Distributed and Replicated Data: Transparency of distributed and replicated data involves hiding of logical and physical structure of data such that user is totally unaware of flow of data between different nodes of the database.

It also provides data independence to the user by providing immunity of user application to change the schema (logical database structure) of the database. Hiding that schema information from user application provides physical data Independence to the application.

Distributed Database has to replicate data among other nodes in the network for performance, reliability and availability reasons. It is off course desirable to have remote data locally for performance point of view.

Fragmentation Transparency: Database is divided in to fragments and moved to different locations. This division is done by partitioning the database in to horizontally, vertically and treat each fragment as a separate database object. This is done for gaining performance, reliability and availability of the database.

Reliability Through Distributed Transactions: Unlike centralized database, distributed databases provides more reliability since they have replicated components. In centralized databases, if the database is down for any reason every node or site of the network will be effected. In distributed databases there is no single point of failure. If one node is down database is still accessible through other nodes in the network due to replication components.

Improved Performance: In distributed databases performance is significantly improved. This performance factor depend upon two main points

- Due to property of fragmentation of the database, so that data is stored at the close proximity to the point of use.
- Due to inherent parallelism of distributed database systems, each transaction or query to the database result in executed in parallel to different nodes in the network. Query and transactions are also divided in to sub query so that many queries can be executed at the same time.

Easier System Expansion: Another big advantage of distributed databases is that, they can expanded according to the need of the organization. Adding a database node on the network will result in adding processing and storage power to the network.

Adding node to the database will not effect the database in any case.

3.2 Databases:

When choosing database for any application we generally have two options. These options are SQL and NoSQL. Most popular ones in SQL databases are PostgreSQL, MySQL and Oracle. In NoSQL databases there are different categories of databases. In Key-value store there are Riak, Mnesia and others, in document store there are CouchDB and other which are mostly selected by the game programmers or database designers according to there requirements.

Both SQL and NoSQL databases support replication methods that are very useful for scalability and load balancing. For instance SQL, MySQL and PostgreSQL can be configured for master-master replication between two masters [GPGD][MRD].

3.2.1 MySQL

[OMS] MySQL server is one of the most widely used open source database management system nowadays. MySQL falls into the category of relational databases. Relational databases are the databases that divide the data in to small chunk rather them putting them all together in one data source. The data is stored in tables which are referenced with another tables so that accessibility and flexibility is maximized. MySQL was originally designed to handle the requirement of large database faster and consistent then any other database. Keeping in mind the ease of use, security, connectivity, scalability and availability.

Scalability refers to balancing the load of the server among different server attach to the database by adding hardware and processing power. Availability refers to increasing the accessibility of the database and to ensure any hardware and software failure do not effect its availability.

3.2.1.1 Availability and Scalability in MySQL. MySQL uses different techniques to provide availability and scalability in the systems. These techniques are: [HAC]

3.2.1.2 MySQL Replication: This technique provides asynchronous method of replication that can be stop and restart any time. Data is replicated from the master node. Due to the asynchronous nature of replication it would not guarantee that data will be replicated from master node to all the slaves nodes immediately. Hence, this technique is useful based on the nature of the application.

3.2.1.3 MySQL Cluster: This technique provide synchronous method of replication. Unlike replication of master slave replication method it creates a cluster. Data in cluster is replicated among all the nodes in the cluster and accessibility to/from all the nodes is immediately. Main restriction using this technique is, all the nodes in the cluster to be replicated should be within the LAN only i.e this technique does not support geographically located nodes.

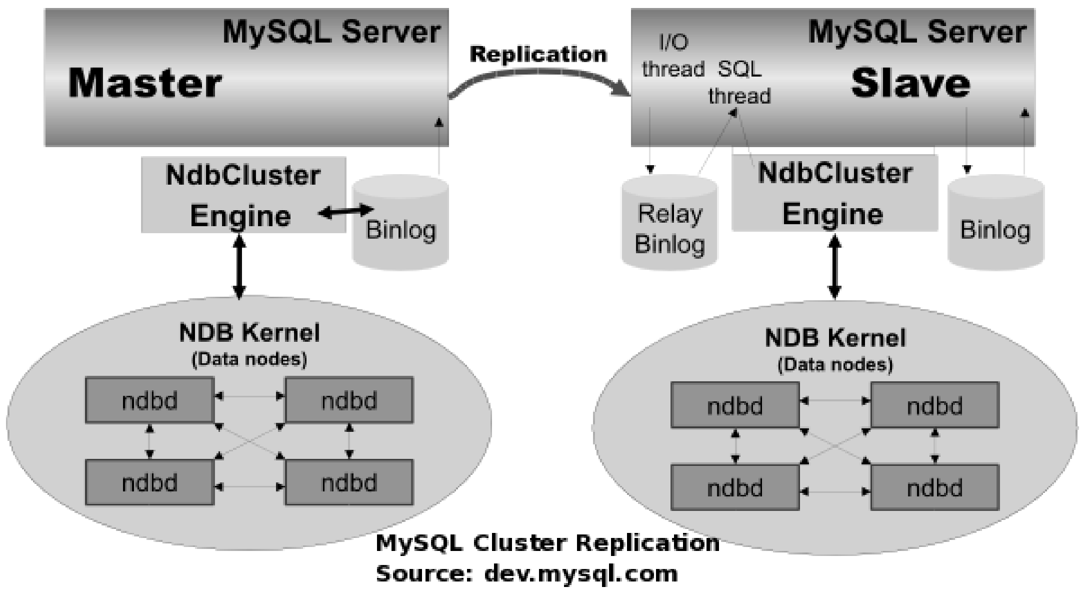

Figure 3: MySQL Cluster Replication

3.2.1.4 DRBD (Distributed Replicated Block Device): It is Linux based third party replicating technique. This technique fulfils all the requirements for high availability of the data. Data is replicated between primary and secondary server by using virtual block device. These virtual block devices are associated with physical block devices. Data written in virtual block devices is replicated among primary and secondary servers physical block devices.

3.2.2 CouchDB:

[Cinfo] Unlike MySQL, CouchDB is a NoSQL schema free database. It is a document store database i.e. data is stored and accessed from the documents of the database. Data in CouchDB is indexed to get high performance and accessibility. CouchDB is famous for web applications and his nature of distributed and ease of use. It provides some robust feature like incremental replication with bi-directional conflict detection, scalability and reporting engine that uses tables and language as a Java-script for querying and indexing.

3.2.3.1 Scalability In CouchDB: CouchDB is also know as peer based distributed database. Because of its creating replicas(copies of the whole database) on each peer connected to the database.

Replication method in CouchDB [Crep] is also synchronous and provide two main methods of replication i.e. Simple and continuous replication. Data is updated or inserted in any node in the network which is selected as a source for the replication (Master Node) and can be replicated to any destination node (Slave Node). One can implement that by posting a HTTP request i.e

```
curl -X POST http://localhost:5984/_replicate -d
'{"source":"http://192.168.0.1/database",
"target":"http://admin:passward@e130.238.11.182:5984/database"}'
```

The above command is for simple replication in CouchDB. This will replicate the source database from source to destination database only once i.e. we have to send request each time we have to replicate. We have to send each HTTP command to each destination node (Slave Node).

Unlike simple replication in CouchDB, in continuous replication we do not have to send request each time we have to replicate i.e. it will continuously detect changes in document and effect changes in all the nodes in the database. The HTTP command for continuous replication is

```
curl -X POST http://127.0.0.1:5984/_replicate -d
'{"source":"db", "target":"db-replica", "continuous":true}'
```

3.2.3 Riak

Riak is a NoSQL key-value store [Bas]. Like CouchDB, Riak also provides web interface and creates replicas of its files distributively. But unlike CouchDB, there is no peer to peer replication but rather do make replicas according to the need of the user. User can change the number of replicas needed among the nodes of the database for faster replication. Riak provides high level of auto balancing to increase availability and performance in large distributed systems. Riak store data in form of Bucket, Key and value. Where bucket is an object of Riak that contains all the information of raw data i.e. all the key and values will be stored in a bucket.

Availability and Scalability in Riak: Riak is very good in providing availability and scalability as compared to MySQL and CouchDB. Riak maintains replication of data among its nodes and is controlled by setting the value called N-Value. This value is default to all nodes in Riak but can be overridden on each bucket.

Riak is purely a distributed system. It is designed to be a distributed system. Reading and writing data and executing map/reduce queries become more faster on adding node to the cluster of the Riak. By Default value of $N = 3$ to all the nodes. Which means Riak will replicate all the data 3 times. Distribution of data is automatically handled by Riak on all the connected nodes.

Comparison Between Riak and CouchDB:

Indexing: Riak do not have built-in indexing technique. RiakSearch provides three way to index the documents [Bas]. Indexing using command line, indexing using Erlang api and index using Solr interface. Indexing make the reads operation speed much faster.

CouchDB uses B-tree [BCou] to index the document and views. Btree is a built-in indexing mechanism in CouchDB.

> ”From a practical point of view, B-trees, therefore, guarantee an access time of less than 10 ms even for extremely large datasets.”[CBT]
>
> *Dr. Rudolf Bayer, inventor of the B-tree*

Btree creates a relation to data on disk through internal sequence identifier for each record. Which make the large dataset accessibility faster.

Accessibility and Erlang as a development language: Both Riak and CouchDB are Erlang based. Hence, inherit all the benefits of Erlang. Riak provide accessibility via HTTP and Protocol buffer but CouchDB is only accessibility over HTTP.

Fault Tolerance, Scalability and Distributed Database: Riak is a distributed database because in order to scale Riak we only have to do is to add another node. Riak automatically adjusts load balancing and replication among the additional nodes. In Riak there is no single point of failure i.e. if any node in the network is down it will not effect the network, other working nodes automatically respond to the client requests. As far as my research is concern, there is no other database like Riak which provide that level of fault-tolerance.

CouchDB is not a distributed system, its a replicating system. Good thing for CouchDB is that you can arrange all kinds of network topology for replication. It does bi-directional replication i.e. any node that is off-line whenever it comes on-line all the changes till the last synchronization will be automatically synchronized. Scalability in CouchDB is very hard to do because you need to implement a sharding layer to replicate your data between multiple Couches.[CDis]

3.3 Description of selected solution:

Pikkotekk is working on their system called lobby system where multiple player can play multiple games at the same time, This system has a requirement of a database, that can fulfil the massive growth of users for there on-line multi player games. Database should provide a powerful, reliable and scalable solution to store massively multi player on-line game data. Due to its massive users, database should provide high level of fault tolerance. Downtime of database effects the whole system.

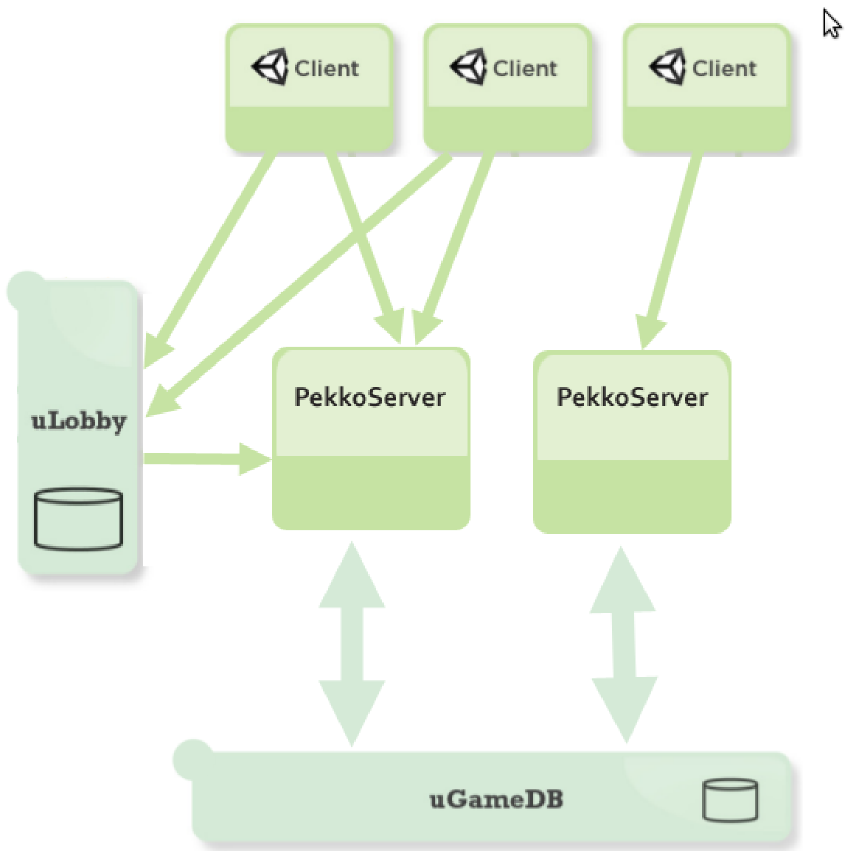

Figure 4: LobbySystem

The above figure shows a general architecture of the lobby system. Where UGameDB is the required database for their system. Unity based games client will either connect to the database through Lobby server for playing games or by directly through the PikkoServers.

In direct comparison, Riak emerges as the best solution for the above mention system, because of its scalable and distributed architecture, it can fulfil all the requirement of lobby system.

4 Riak - A Distributed database for Distributed system.

Now the desired approach has been selected and described briefly, the concepts involved in completeness of this project will be presented here.

The main focus of this chapter will be detailed presentation of Riak database. [Bas] Riak comes with open source/enterprise editions influenced by Dr.Eric Brewers CAP theorem and Amazons dynamo paper. Riak is build for ease of operations and fault tolerance. Riak is distributed under the license of Apache 2 open source. Riak is written primarily in Erlang and C as a programming language. Since Riak is written in Erlang, therefore excel in fault tolerance, parallelism and stability. Main feature includes de-centralized key-value store, flexible Map/Reduce engine, fault tolerance, highly scalable, friendly HTTP/JSON Query interface that is best suited for web applications.

This chapter is divided in two parts, first part explains the basic theoretical principles of Riak. The second part will explain Riak from developers aspects.

4.1 Data Storage in Riak:

Riak supports multiple back-end data storage. Bitcask is a default database for Riak and can support plug able backends data storage like Dets, Ets, Erlang balanced trees (gb_trees) and writing directly to the filesystems.

Data in Riak can only be organized in bucket, key and value pair format. Each values in a bucket is identified by a unique key i.e. each bucket contain a unique pair of key value pairs. Each bucket/key entries can point to another entries to form a link of entries and these entries can be accessed through link walking via Riaks HTTP interface.

4.2 Client Libraries in Riak:

Currently Riak supports six client libraries that can be used to interface with Riak. User can select any language among the supported client libraries. Selection of libraries is also depend upon the application and programmer ease. The supported languages are:

Erlang: Since Erlang is a development language of Riak, it is tightly integrated with HTTP and protocol buffer interfaces of Riak.

JavaScript Riak used JQuery in order to interface to Riak. JavaScript is an official language for querying Riak.

Java Support Java language for interfacing. Client can integrate Java based application by using Java Client for communicating Riak.

PHP Can interface with PHP language using PHP client.

Python: Can interface with Python language using Python client.

Ruby: Ruby is also officially supported language for Riak interfacing.

4.3 Clustering in Riak:

Nodes in Riak form a cluster. This cluster is divided in to Virtual Nodes(Vnodes) and partitions to form a ring to get all the advantages of Riak. Ring is a 160 bit integer space divided in to equally sized partition[Bas].

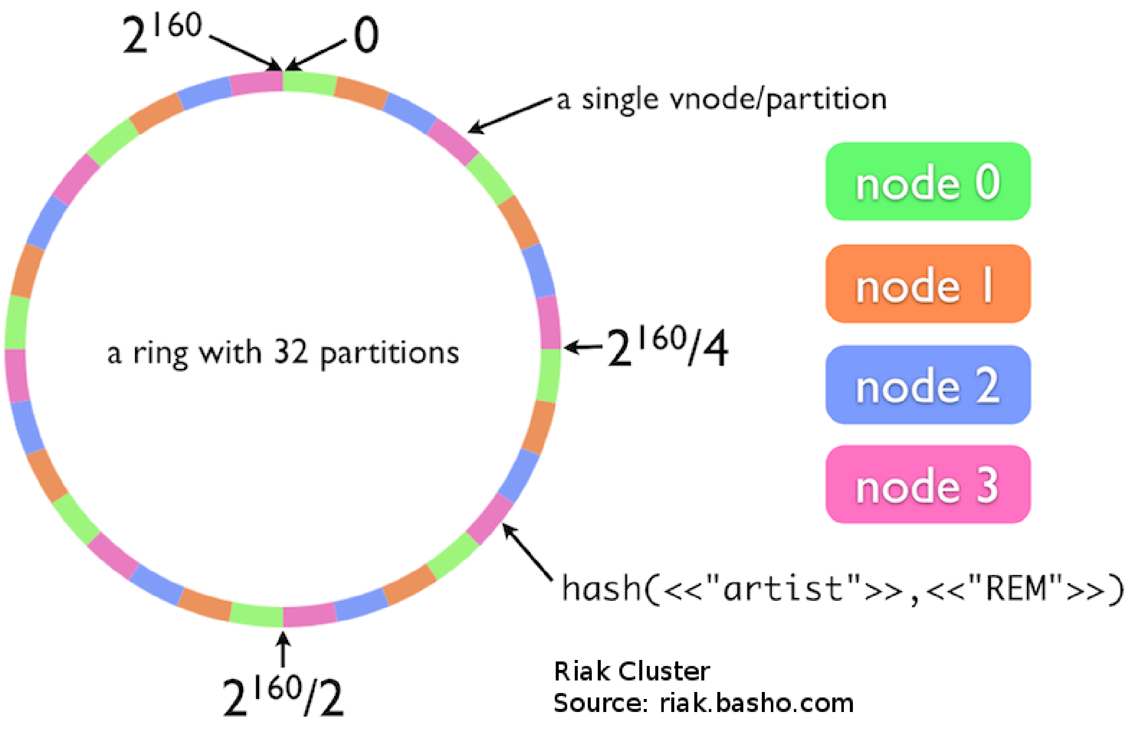

Figure 5: RiakCluster

Each node in a ring also known as physical node runs a certain amount of virtual nodes(VNodes). Each Vnode occupy a single partition in the ring. Partition size of the ring is defined when Riak is configured or at cluster initialization. To understand more clearly how Riak form a ring cluster, consider partition size of ring is 32 as shown in above figure. We calculate the number of Vnodes as

```
Vnode = (number of partition) / (number of nodes)
```

Each node in a cluster is responsible for 1 / (total number of physical nodes) i.e. if we have 4 nodes then each nodes is responsible for 8 partitions in ring as shown in above figure.

This shows the followings [Bas]

- There is no master node all nodes in Riak are serving equal purpose.
- Riak uses consistent hashing to distribute data around the cluster.
- Each Node in cluster is capable to serve each client requests.

Moreover, Riak cluster is capable of shrinking and expanding dynamically that is if any node is added or removed from the cluster. Cluster expands or compact it self to distribute the data accordingly.

4.4 The CAP Theorem:

CAP Theorem was formulate by Professor Brewer from University of California. CAP stands for Consistency, Availability and Partition tolerance. In distributed system, Partition tolerance also known as fault tolerance is considered as a most important property [CThe1]. Partition tolerance is a property that implies subsystem should not respond incorrectly unless whole system failure. If any component of the system is not functional or have some kind of software or hardware issues then operation should be carried out successful. Consistency means in distributed system operation carried out at one node should have consistent state through out all the nodes in the system unless explicitly changed. The operation should be carried out as a whole or should not carried out at all. Availability means system should respond to all client request. There should not be any case where system fail to respond client request. According to [CThe1][CThe2][CThe3] [CThe4] the CAP theorem is

> It states, that though its desirable to have Consistency, High-Availability and Partition-tolerance in every system, unfortunately no system can achieve all three at the same time.

4.4.1 SQL, NoSQL Databases Vs CAP theorem:

According to [Bre00] theoretically CAP is required for a distributed systems but in reality only two properties can be follow. The relational databases like MySQL and Postgres follow consistency and availability. Since the standard follow by relational databases are ACID. ACID stands for Atomicity, Consistency, Isolation and Durability. NoSQL databases like MongoDB, Terrastore, Scalaris, Berkeley DB, MemcacheDB and Redis follow Consistency and partition tolerance. Whereas Cassandra, CouchDB and Riak follow Availability and Partition tolerance [VSNO].

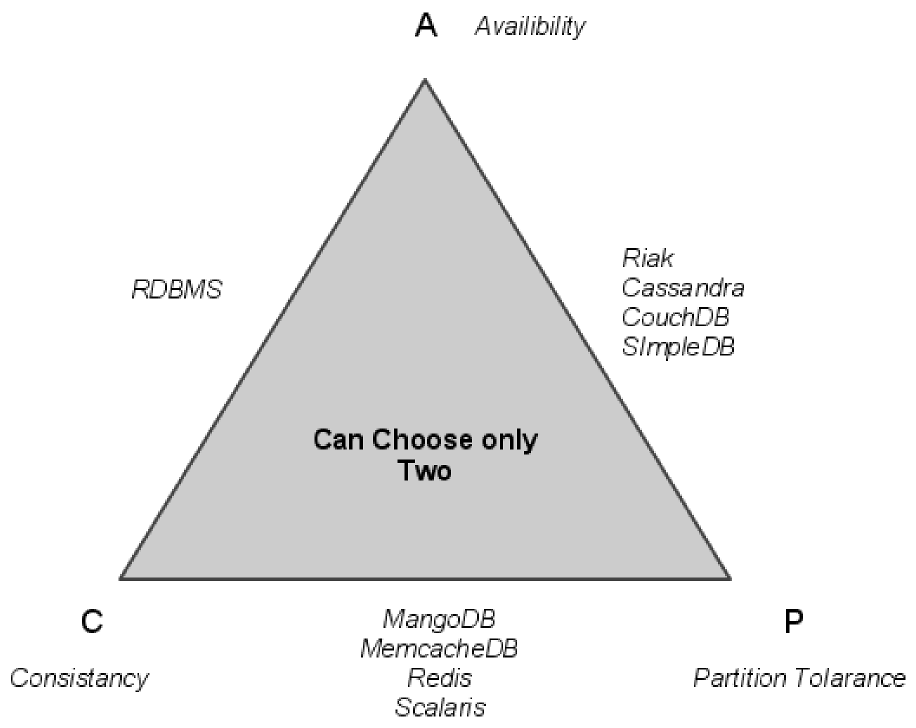

Figure 6: CAP

Riak gives option to the developer to control CAP by giving the feature of changing the value of N, R and W. Changing these values means tuning Riak by telling how many copies of data you want at bucket level. This feature makes Riak differ from other NoSQL databases. Riak also provides eventually consistency next to availability and partition tolerance. Eventually consistency means, that database might be inconsistent for a very short time. For Riak, eventually consistence is in term of millisecond [TCR].

4.5 Replication In Riak:

Data in Riak is replicated according to the default value of N that is N=3. Which means data in cluster will be replicated among 3 nodes in the cluster. It is recommended that there should be 3 nodes in a Riak cluster to get maximum performance from Riak. One can have three or more number of nodes on same machine, but this will not result in better performance because we are replicating some data on same machine i.e. each data will have three copies on same machine. Adding physical node in the Riak cluster will result in sharing load and processing.

To change the N-value for a bucket from default (N=3) to some thing else one can issue a Put command.

```
$ curl -v -X PUT -H "Content-Type: application/json" -d '{"props":{"n_val":2}}'
http://127.0.0.1:8098/riaksearch/player
```

This command will change the value of N from 3 to 2 i.e. we want Riak to make copies of each data in bucket 2 times in a cluster.

4.5.1 R Value and Read Fault Tolerance:

For read fault tolerance purpose, Riak allows users to use R value for fetching(reading) data from the bucket. R value make sure that number of nodes should respond with the result for the read request to consider successful.

```
riakc_pb_socket:get(Client, Bucket, Key,[{r, 1}])
```

In above GET command for getting value from bucket ¨Bucket¨ of key ¨Key¨ from open socket ¨Client¨ will send request to all nodes in the cluster and get command will be considered sucessful if any one node respond with the result. This R value improve read fault tolerance in Riak.

4.5.2 W Value and Write Fault Tolerance:

For write fault tolerance purpose, Riak allow users to use W value to Write/Update data in to the bucket. W value make sure that number of nodes should result in a successful write for a request to consider sucessful.

```
riakc_pb_socket:put(Client,InsertObject, [{w, 2}])
```

The above PUT command will insert Object stored in variable ¨InsertObject¨ with W value 2. Which means objects will be inserted according to value of N but this request will be considered successful only if 2 nodes in cluster will respond successful write.

4.5.3 Conflict Handling in Riak:

Since Riak is a distributed database, many users can commit a transaction at the same time to the same or different nodes. Although, due to high availability in Riak, all nodes in Riak cluster are not required to respond to client request any single node in cluster is fully capable to respond to client request. Multiple clients can update same attribute at the same time this can cause conflict in database. Different NoSQL databases use different techniques to solve conflicts in distributed database.

Cassandra uses time stamps to tag the data and compare the time stamps to determine the newest version of the data.

MongoDB uses last one wins techniques to resolve conflicting issues [RVN].

Neo4j uses traditional RDBMS techniques to solve conflicts in database. Unlike Riak which ensure that database is always available for datastorage, Neo4j do not allow conflict to happen on the first place and do necessary precautions to prevent from happening. These techniques are describe in [NeoTrans].

In Riak for solving conflict issues vector clock is used. To allow multiple users to change same attribute of a bucket at the same time. We need to enable allow_mult property to true. By default this property is false.

```
$ curl -v -X PUT -H "Content-Type: application/json"
-d '{"props":{"allow_mult":true}}' \
 http://127.0.0.1:8098/riak/Bucket_name
```

this command will enable allow_mult property to true for the bucket named ¨Bucket_name¨. Vector clock keeps track of all the versions of data on the bucket where allow_mult property is true. When a value is stored in Riak, vector clock initialize a first version of the vector clock. When another user or a same user wants to change the same data, he will provide the previous updated vector clock of the object in order to change the current object. Changes applied by the next user issue another vector clock for that object. In this way, Riak resolves the conflict problems.

4.6 HTTP Interface:

Riak uses REST API (Application Programming Interface) for accessing object from the cluster. Riak uses HTTP GETs, PUTs or POSTs for the storage operation and reading link walking and MapReduce. Applications that uses Rest as their API can access everything via HTTP interface like images, videos, audios in short every kind of data.

4.7 Components of Riak:

The latest release of Riak comes with the following components.

1. Riak Core - Dynamo-Inspired distributed systems framework.
2. Riak KV - Distributed Key/Value store inspired by Amazons Dynamo.
3. Riak Search - Distributed index and full-text search engine.

Riak search is a sub component of Riak KV. If you have Riak search install and running then you also have RiakKV cluster. Which means, by using Riak search you can get advantages of both Riak KV and Riak Search.

4.7.1 RiakSearch:

RiakSearch is build around RiakCore and tightly integrated with RiakKV [RSea]. Due to these factor, RiakSearch inherit advantages of distributed, scalable, fault tolerance and real time from Riak KV. RiakSearch provides features which includes full text search engine facility and built-in indexing of data.

RiakSearch provide build-in indexing service of data that makes the reading and writing the data in to the database much faster.

4.7.1.1 Indexing In RiakSearch: For indexing documents in RiakSearch, system use a schema file in which we define a required fields, unique keys and the default analyser for each field. Each bucket have its individual file.

RiakSearch tokenizes the schema file or document in to an inverted index using standard Lucene Analayzer [RSea]. When the document is inverted index RiakSearch uses consistent hashing to distribute the indexed data across the cluster called term partitioning. According to [TPI] by using term partitioning in distributed databases increase the overall throughput of the query with large data set.

4.7.1.2 Querying in RiakSearch: Querying in RiakSearch is done by following lucene syntax. Lucene query has an advantage of full-text search engine. Lucene query syntax support term searches, lexicograph-ical range queries, field searches and wild card searches.

Querying any database involve two steps which are planning and executing. In planning database tries to maximize data locality and inter-node traffic. In executing step database tries to find the nature of the query. If the query is single term query then it will be executed on single node. Where as wild card queries and range queries are executed in multiple nodes.

RiakSearch uses a series of merge-joins, merge-intersections and filters to generate the resulting set of matching bucket/key pairs[RSea].

5 Implementation Environment:

5.1 Erlang:

Erlang was develop in 1980 by company named Ericsson. Erlang is short abbreviation of Ericsson Language. The history behind Erlang is, group of person were doing a research on some series of programming language to find the best suited language for telecommunication. Unfortunately they did not find the best match, so they end up by implementing there own language known as Erlang.

Erlang was developed to satisfy the highly concurrent need of Telecom applications. Moreover, it is also good at satisfying the need of multiple concurrent processes, distributed applications, fault tolerance and ability of handling large amount of concurrent transactions. Erlang is considered as one of the best programming language for distributed systems. Jeo Armstrong one of the main developer of Erlang release the first version of Erlang in 1986 at Ericsson Computer Science Laboratory. Since then continuous progress has been made up till now to make Erlang one of the best functional language for telecommunication.

In 1996 Ericsson released an API known as OTP (Open Telecom Platform) in Erlang. OTP offered lot of testing, error handling mechanism and techniques for system robustness. Erlang become officially open sourced for public in 1998.

5.2 Looby Server:

Looby server is distributed system that is implemented in Erlang/OTP at PikkoTekk. Below figure shows a general framework of Looby Server.

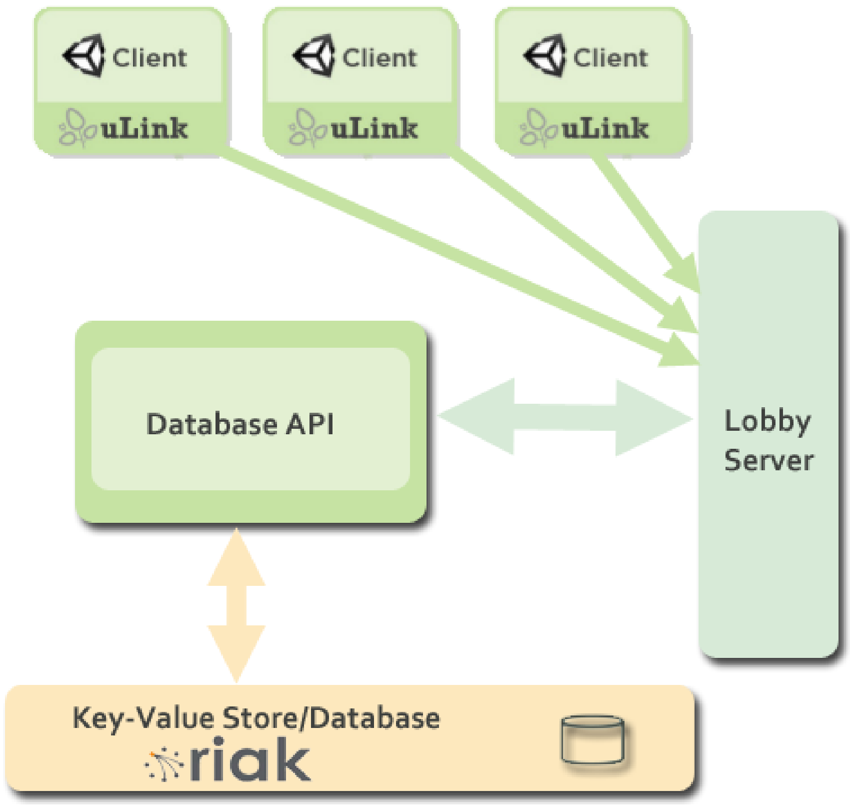

Figure 7: Lobby Interface

the bottom layer in the figure 7 is database layer. This layer deals with the database operations which includes all the inventory information, user statistics, player information and games information at the lobby server. User connects to the lobby server by giving there credential information. After successful logon, user have to create an instance of the game he/she wants to play. Lobby server have predefined list of games that player can choose and can create instance of each game. Users can also include the game of its own choice on the server to be played among his peers. Since lobby system is a multi player application, so multi user can logon and can join any game they want. Each player in the system can play many games at the same time or can create instance of more them one game at the same time.

Whenever game want to get or save data to/from the database it will communicate with database API. Database API responsibility is to communicate with database and in response from the database, data is forward to the database API. Database API inform the result of communication with the database to the game.

5.3 Unity:

[U3d]Unity is a game development tool. Unity provides simple IDE for designing and developing 3d games faster and easily. JavaScript and C#.Net is main game development language support by unity. Game developer can use unity on MAC and Windows platforms.

The feature provides by unity are

- Integrated Editor
- Component based Architecture
- Game Engine
- Scripting Platform

Unity provides an integrated editor where user can organize there games objects, games assets, games scripts in hierarchy order. It contains very powerful interface where one can drag and drop games objects on to the game and can get live view of the game in development mode. Switching from development mode to play mode only started moving object and add animation to it. This feature is very helpful for the game developer.

Game engine in Unity contains all the necessary tools that are required to develop games not only for browsers also for portable devices. such as

- Graphics Engine (For 3d graphics purposes etc)
- Physics Engine (For changing object movement etc)
- Audio Engine (Engine that can play audio in 2D and 3D space)
- Animation systems (for mixing and real time vertex/bone reassignment etc)

Provide Scripting platform that embed mono (Software that allows developer to programme any language on any platform or that makes languages platform independent). Support scripting language include c#, JavaScript and boo [Ubas]

5.4 Database Design:

The requirement of distributed database is to handle the massive amount of games data from massive online multi players. Game data includes player information, game information, which player is playing which game, which player is playing against which player, group of players, group against which group of players, score of each player on each game and inventory information of each player.

Schema file for Riak is as fellows

```
{
    schema,
    [
       {version, "1.1"},
       {default_field, "Nickname"},
       {default_op, "or"},
       {n_val, 3},
       {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
    ],
    [
       %% Don't parse the field, treat it as a single token.
  {field, [
           {name, "fname"},
           {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
       ]},
       %% Don't parse the field, treat it as a single token.
  {field, [
           {name, "lname"},
           {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
       ]},
       %% Don't parse the field, treat it as a single token.
  {field, [
           {name, "Nickname"},
           {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
       ]},
           %% Don't parse the field, treat it as a single token.
  {field, [
           {name, "InstanceID"},
           {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
       ]},
           %% Don't parse the field, treat it as a single token.
   {field, [
           {name, "gameID"},
           {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
```

```
        ]},
        %% Parse the field in preparation for full-text searching.
    {field, [
            {name, "qname"},
            {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
        ]},
    %% Parse the field in preparation for full-text searching.
    {field, [
            {name, "itemname"},
            {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
        ]},
    %% Parse the field in preparation for full-text searching.
    {field, [
            {name, "itemvalue"},
            {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
        ]},
    %% Parse the field in preparation for full-text searching.
    {field, [
            {name, "competitorName"},
            {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
        ]},
    %% Parse the field in preparation for full-text searching.
    {field, [
            {name, "score"},
            {type, integer},
            {padding_size, 10},
            {analyzer_factory, {erlang, text_analyzers, integer_analyzer_factory}}
        ]},
        %% Treat the field as a date, which currently uses noop_analyzer_factory.
    {field, [
            {name, "date"},
            {type, date},
        {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
        ]},
        %% A dynamic field. Anything ending in "_text" will use the standard_analyzer_facto
    {dynamic_field, [
            {name, "*_text"},
             {analyzer_factory, {erlang, text_analyzers, standard_analyzer_factory}}
        ]},
       %% A dynamic field. Catches any remaining fields in the
       %% document, and uses the analyzer_factory setting defined
       %% above for the schema.
     {dynamic_field, [
             {name, "*"}
        ]}
     ]
}.
```

The above schema shows the schema file for game database. Any bucket on which this schema file is loaded will automatically index the fields according to fields name. These fields are player first name(fname), player last name(lname), PlayerNickname, Game Instance ID(InstanceID), Game ID (gameID), quest name (qname), Weapon ID (Itemname), Weapon stat (Itemvalue), Player Score (score), date and time.

5.5 Interfacing With Riak Database:

Since Unity and Riak both are platform dependent. Unity can only run under windows or MAC platform and Riak do not support windows and currently supports only Linux platform. Keeping these limitation in mind, there is a need of APIs that can interact between game and database.

So the proposed architecture is:

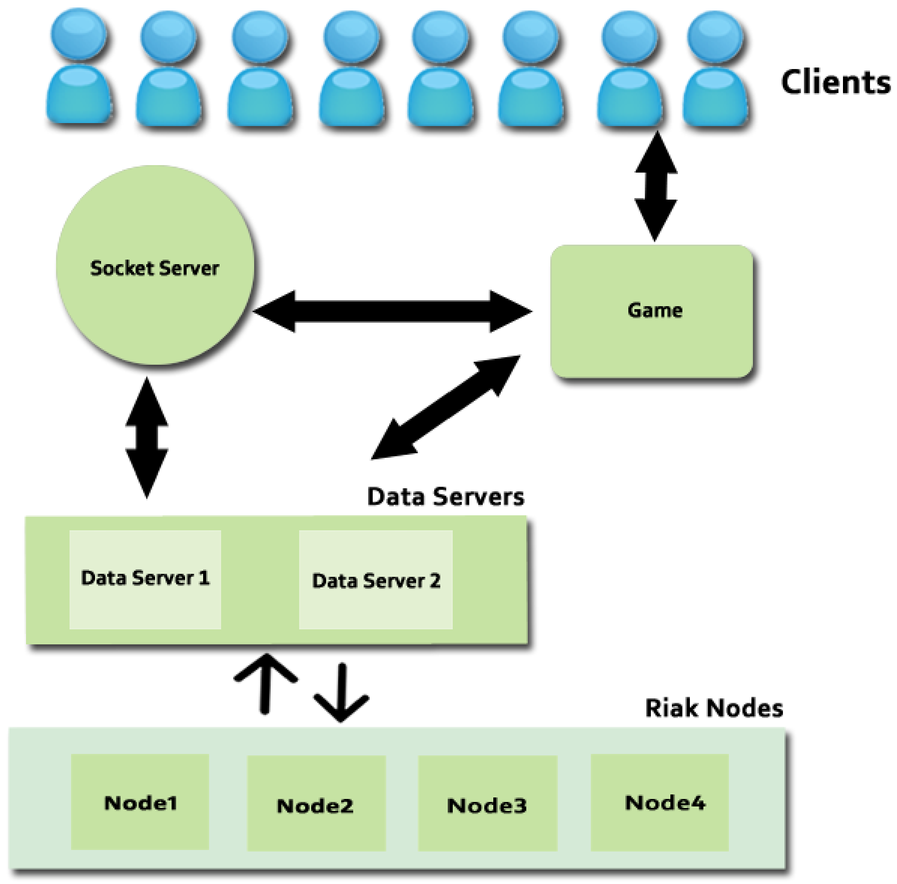

Figure 8: Interfaces

Lobby system have lot of games and each games have many players. Whenever, game need data from the database. Game will try to connect to the socket server and request IP address and port of data server from where it can get data. Socket server is connected with data servers and have list of available and working data servers. If any data server is up or down socket server is informed about the current status of each data server. So socket server respond game request by providing the running and feasible data server IP and port address. Having IP and port of the data server, now game can send database operations or query request to the database servers. Database server are connected to Riak nodes. Riak can respond to any request from data server distributively.

5.5.1 Socket Server Interface:

Most of the socket server API functions are written in socketserver.erl.

The socket server interface was designed to enable communication between Linux(Riak) and Windows(Unity Games). In Linux node, there is always a socket running named "Socketserver". The functionalities of the socket server are.

- Provide Dynamic Port functionality
- Provide Multi Socket support

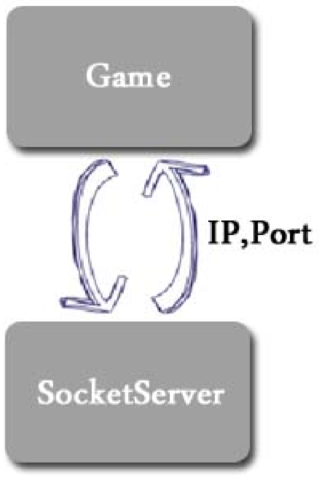

Figure 9: Socket Server Interface

Each game will be assigned a dynamic port so that each game can get individual port. Dynamic port is very effective because sending a receiving massive data to/from one port and cause problem in long terms.

Multi socket is very helpful for fault tolerance purposes. Putting load on one socket can be cause of bottleneck and abnormal terminations. Techniques are applied so that load sharing between the data server can be handled.

5.5.2 Data Servers:

Most of the data server functions are written in module erl_api.erl

Data server is an API that is used to communicate with Riak directly. It contains functions that are responsible for doing database operations on the database.

Since Riak is key-bucket store so the data should be stored in form of key bucket value. By keeping this in mind, I am populating data in Riak in form of

```
Bucket, Key,[{field1,value1},{field2,value2},{field3,Value3}....
```

In this case we have unique key and each key contain values of all the attributes of the schema shown in 5.5.

5.5.2.1 List_buckets(Riak_Client_Object): This function is used to fetch the list of all the buckets in the database. Riak_Client_Object is an object of Riak client, in our case I am using Erlang as Riak Client.

```
List_buckets(Client).
```

This function call will return all the bucket from Riak object Client.

5.5.2.2 List_keys(Bucket,Riak_Client_Object): This function is used to fetch all the keys that a bucket ¨Bucket¨ contains. Where B̈ucketänd R̈iak_Client_Objectäre two parameters required for that function. B̈uckets̈hould be name of the bucket from where the list of keys are required.

```
List_keys(Player, Client)
```

This function call will return all the keys from bucket P̈layer¨.

5.5.2.3 Get_bucket_values(Bucket,Key,Riak_Client_Object): This function is used to fetch all the values from the bucket. Where Bucket, Key and Riak_Client_Object are the two parameters required for that function. Since each key in bucket contain all values of the attributes of the schema so this function will return values of specified key K̈ey¨. For example

```
get_bucket_values(Player, yousaf, Client).
```

This function call will return values with there attributes from bucket ¨Player¨ of Key ¨Yousaf¨.

5.5.2.4 Search_key(Bucket, Query,Riak_Client_Object): This function will query the database for the specific query and return the number of keys that fulfil the query condition. Where Bucket, Query and Riak_Client_Object are the parameter required for the function. This function can also return result for query of types wildcards and range. Query parameter needs the desired indexed query.

```
search_key(Player, date:[20110501 TO 20110601], Client).
```

This function call will return the key from bucket ¨Player¨ that are generated from date 2̈0110501¨ to 2̈0110601¨.

5.5.2.5 Insert_data(Bucket,Key,Data,Riak_Client_Object) This function is used to insert data in to the database. Where Bucket, Key, Data and Riak_client_Object are the parameters that are required for this function. Data should be in format of list with tuples.

```
insert_data(Player, test, [{playername, test1},{score, 0}, {data, 201106026},{gameID
, snowbox}], Client).
```

This function call will insert one record into the database of bucket ¨Player¨ with key ẗest.

5.5.2.6 Delete_key(Bucket,Key,Riak_Client_Object): This function is used to delete any key in the bucket. Where Bucket, Key, Raik_Client_Object are the required parameters of the function. Key to be deleted has to be passed to the parameter ¨Key¨

```
delete_key(Player,yousaf,Client)
```

This function call will delete the key ¨yousaf¨ from bucket P̈layer.̈

5.5.2.7 Update_value(Bucket,Key,Field,Value,Client): This function is used to update the current value of any key in the database. Where required parameter are Bucket, Key, Field, Value, Riak_Object. Field is attribute of the bucket whose value is required to update. Value is the updated value.

```
update_value(Player, yousaf, score, 200,Client)
```

This function call will update the value of attribute ¨score¨ to 200 from key ¨yousaf¨ of bucket P̈layer¨.

5.5.2.8 Search_by_field(Bucket,Key,Field,Client): This function returns value of specified attribute. Where required parameter are Bucket, Key, Field, Riak_Object. Field is the parameter, where the required attribute name is passed to fetch the value from the database.

```
Search_by_field(Player,yousaf,playername,test1).
```

this function call will search for the player name whose value is test1 and return the complete record of it.

5.5.3 C# Interface:

Function of this interface is to provide user friendly environment for the game developer. There are two class designed named Bucket.cs and Database.cs. These two class provides ease to the programmer to work around the database.

Database class is a singleton class to connect to the Riak database. This class contains all the methods required for the connection to the database. To connect to the database programmer has to create an instance of that class. Object created by this instance can be used any where in the class.

```
Database b = Database.getInstance;
b.connect(MasterSocketIP,MasterSocketPort);
```

Since Riak is a bucket key-value store, Bucket.cs is designed so that programmer can create instance of any bucket he want to use. Instance of any bucket created by this class can be used any where in the class.

Benefit of this class is, programmer do not have to specify bucket name again and again for any database operation. for example

```
Bucket a = new Bucket(b,"player");
```

This class required two parameters. Database Object and name of the Bucket. This class act as a dictionary for the programmer like game developer can add, update and delete objects from bucket.

```
a.add("peter","playername","Peter")
a.delete("peter")
a.update("yousaf","score","300")
```

6 Testing and Evaluation:

This chapter introduces some techniques to test and evaluate the database.

6.1 Testing by Integrating with The Game:

PikkoTekk develop a test game for testing the overall system known as SnowBox. The sequence diagram for some database operations that involves all the modules involved in the process is described in the figure below

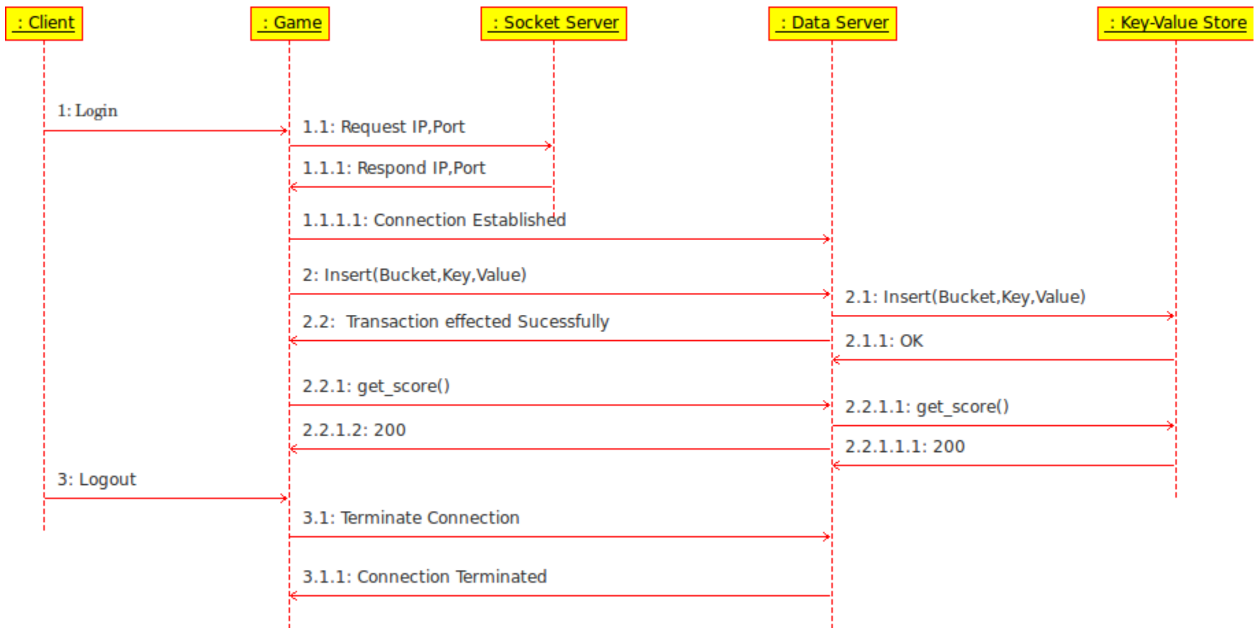

Figure 10: Sequence Diagram of DB transactions

The figure 10 shows a use case of user to login into the system, does set of transactions on the database and logout to the system. The live demonstration of the game is shown in the figure below

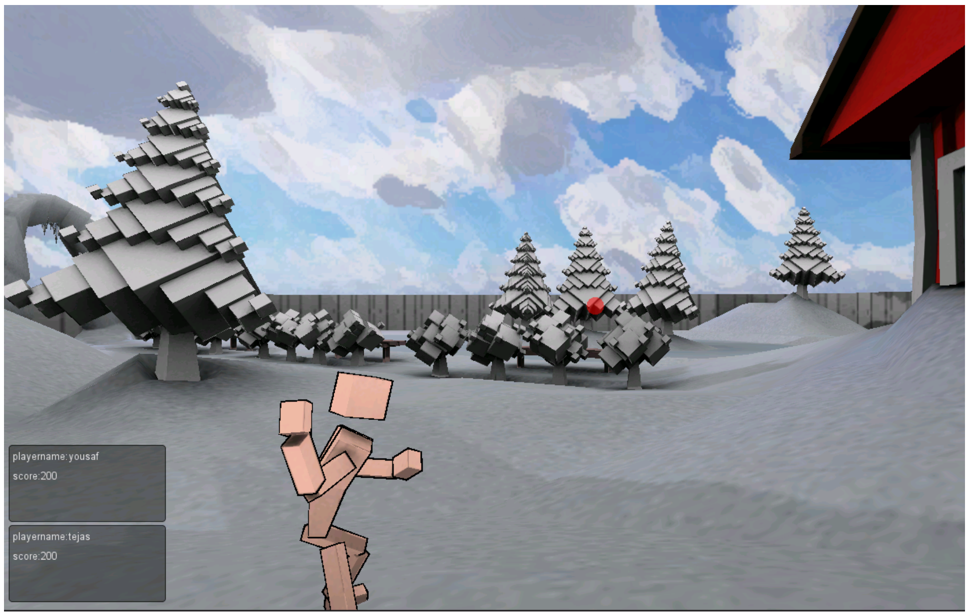

Figure 11: Snowbox Game

6.2 System Performance Evaluation:

The process to evaluate the performance of the database is carried out by executing a set of transaction that includes the get/select and puts/insert/update on the database via database API. One of the requirements of Pikkotekk is to evaluate the performance of Riak, CouchDb and Mysql. For this purpose separate API is designed to communicate with each database separately. Each API executes a fix number of transactions on the database.

The file riaktest.erl, mysqltest.erl and couchtest.erl are used. Each file contains a group of API functions which are required to test the performance of the database such as creating a unique key to be inserted each time the put transaction is applied, getting a set of keys/buckets or records from the database.

Databases are evaluated by the number of transactions per seconds and maximum number of user can be connected to the database on the fixed number of time. For this purpose I choose Tsung. Tsung is an open source distributed load testing and performance evaluation tools for client server applications. All three database are executed under tsung with the parameters of 200 users that runs parallel to execute the transactions to be hit on the database in the duration of 1 minutes.

6.2.1 CouchDB Vs MySQL VS Riak Under 1 node Cluster:

The comparison between CouchDB, MySQL and Riak is:

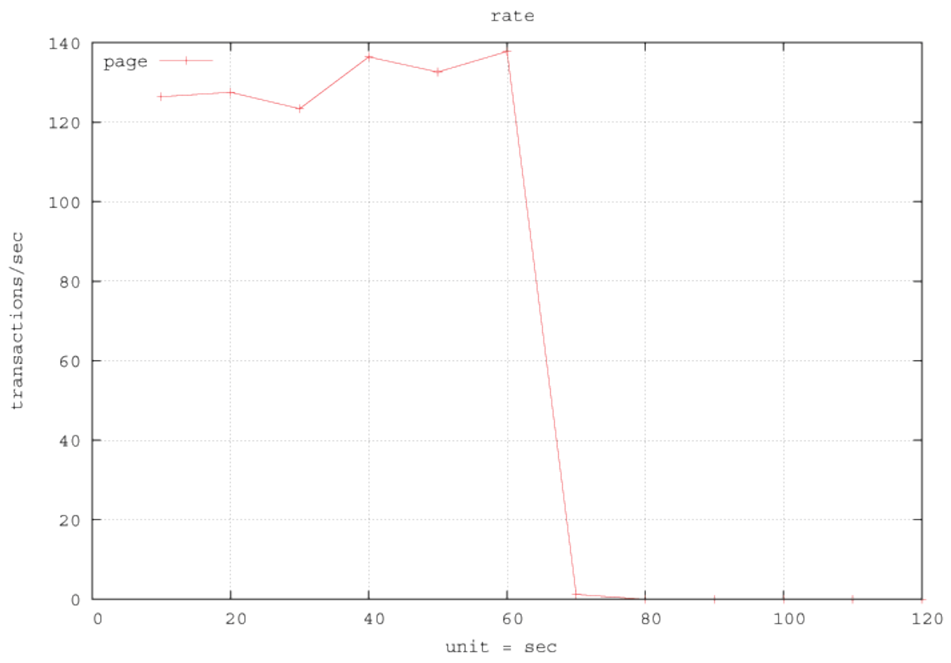

Figure 12: CouchDB TransactionRate

Figure 12 shows that, by concurrently executing 200 users CouchDB manage to execute maximum of 140 transactions per seconds in the duration of 1 min.

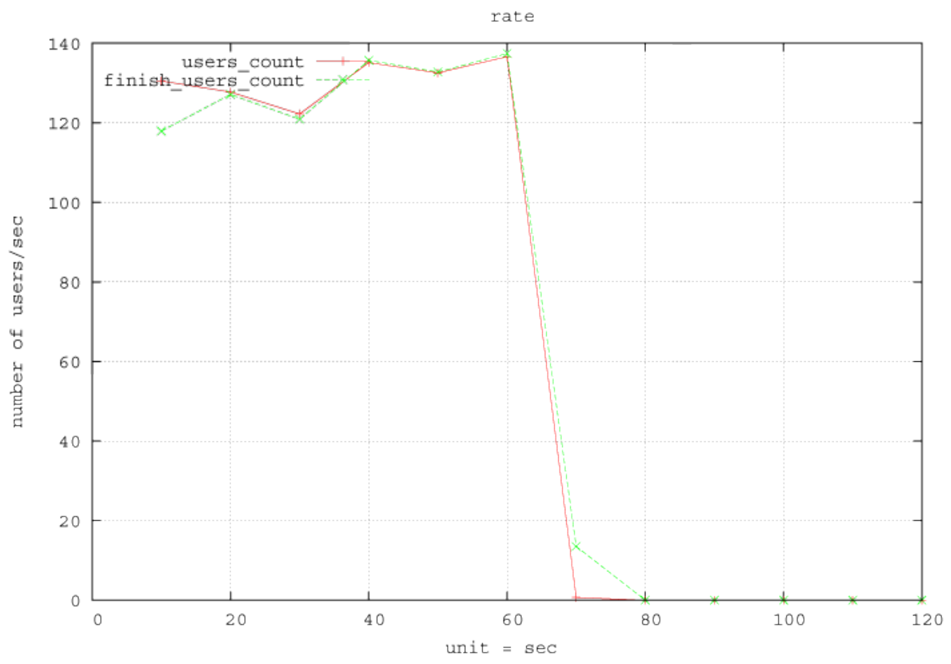

Figure 13: CouchDB Users Arrival Rate

Figure 13 show that, after approx 10 sec tsung manage to send approx 130 user connections request out of which CouchDB able to connect only approx 120 users per second successfully at time duration of 1 minutes CouchDB connects 140 users per seconds.

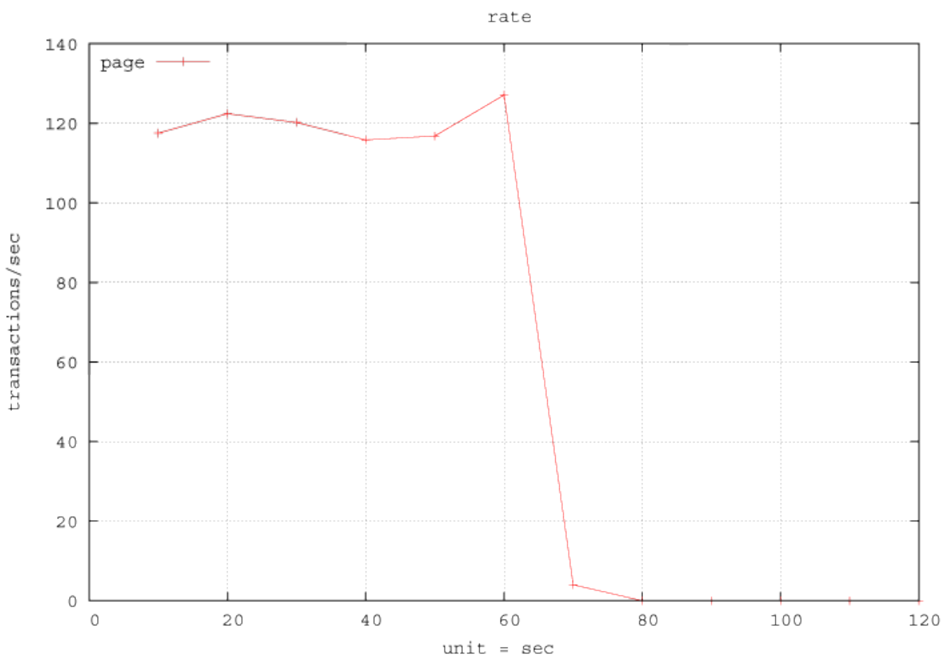

Figure 14: MySQL TransactionRate

Figure 14 shows that, by concurrently executing 200 users MySQL manage to execute maximum of approx 130 transactions per seconds in the duration of 1 min.

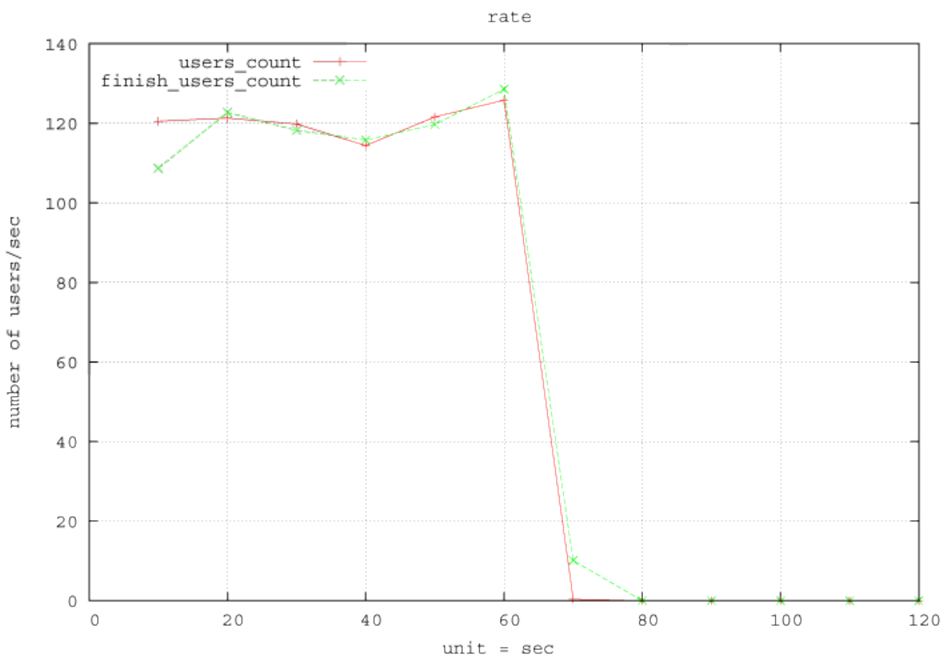

Figure 15: MySQL User Arrival Rate

Figure 15 show that, after approx 10 sec tsung manage to send approx 120 user connections request out of which MySQL able to connect only approx 110 users per second successfully and at time duration of 1 minutes MySQL connects approx 130 users per seconds.

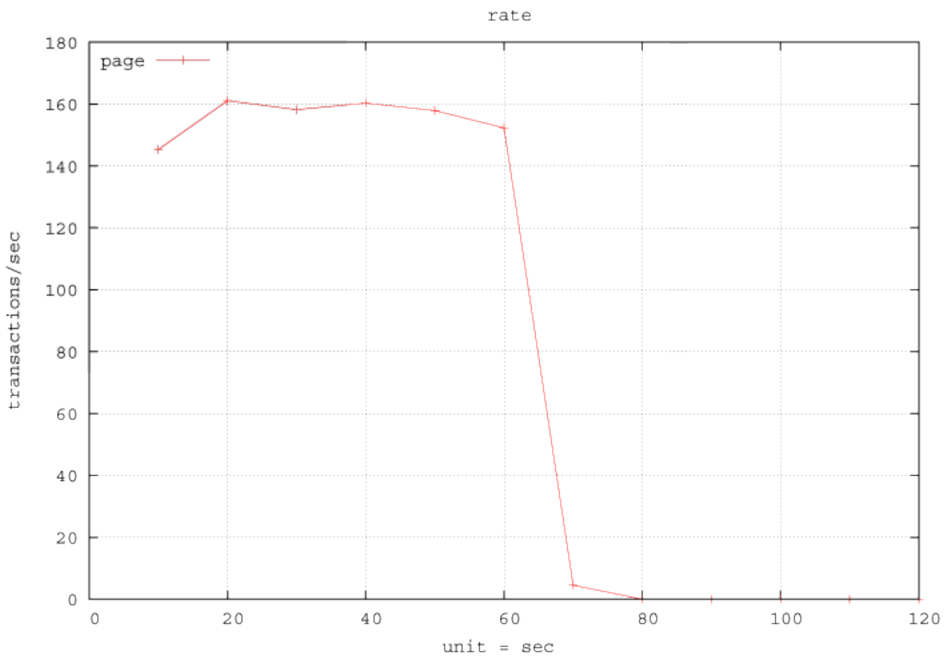

Figure 16: Riak Transaction Rate

Figure 16 shows that, by concurrently executing 200 users per second Riak manage to execute maximum of 160 transactions per seconds in the duration of 1 min.

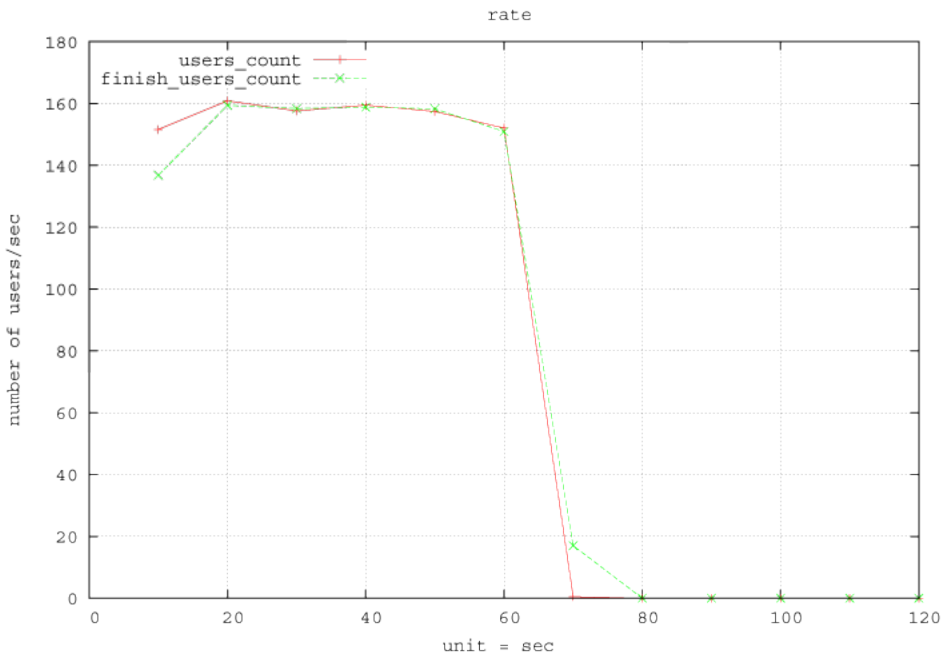

Figure 17: Riak Users Arrival

Figure 17 show that, after approx 10 sec tsung manage to send approx 150 user connections request out of which Riak able to connect only approx 133 users per second successfully and at time duration of 1 minutes. Riak connects approx 160 users per seconds.

From above comparison, it is concluded that Riak performs better in comparison to CouchDB and MySQL.

6.2.2 Comparison Between 1 and 2 Riak Nodes:

To get the maximum numbers of transaction per seconds and users per seconds parameter for evaluation are changed. Previously maximum users are based at 200 users per sec and total time duration was 1 minute but now users are increased to 5500 and total time will be 5 minutes. These parameter are defined on tsung.xml.

```
<?xml version="1.0"?>
<!DOCTYPE tsung SYSTEM "/usr/local/share/tsung/tsung-1.0.dtd">
<tsung loglevel="notice" version="1.0">
<clients>
<client host="localhost" use_controller_vm="true" weight="5" maxusers="100000"/>
</clients>
<servers>
  <server host="130.238.11.182" port="8098" type="tcp"/>
  <server host="130.238.11.167" port="8098" type="tcp"/>
</servers>
<monitoring>
    <monitor host="myserver" type="snmp"/>
</monitoring>
<load>
        <arrivalphase phase="1" duration="27" unit="second">
        <users arrivalrate="455" unit="second"/>
        </arrivalphase>

        <arrivalphase phase="2" duration="27" unit="second">
        <users arrivalrate="910" unit="second"/>
        </arrivalphase>
```

```
<arrivalphase phase="3" duration="28" unit="second">
<users arrivalrate="1365" unit="second"/>
</arrivalphase>
<arrivalphase phase="4" duration="27" unit="second">
<users arrivalrate="1820" unit="second"/>
</arrivalphase>

<arrivalphase phase="5" duration="28" unit="second">
<users arrivalrate="2275" unit="second"/>
</arrivalphase>

<arrivalphase phase="6" duration="28" unit="second">
<users arrivalrate="2730" unit="second"/>
</arrivalphase>

<arrivalphase phase="7" duration="27" unit="second">
<users arrivalrate="3185" unit="second"/>
</arrivalphase>

<arrivalphase phase="8" duration="27" unit="second">
<users arrivalrate="3640" unit="second"/>
</arrivalphase>

<arrivalphase phase="9" duration="27" unit="second">
<users arrivalrate="4095" unit="second"/>
</arrivalphase>

<arrivalphase phase="10" duration="27" unit="second">
<users arrivalrate="4550" unit="second"/>
</arrivalphase>

<arrivalphase phase="11" duration="28" unit="second">
<users arrivalrate="5500" unit="second"/>
</arrivalphase>
```

```
</load>
<options>
   <option type="ts_http" name="user_agent">
    <user_agent probability="80">Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.8) Gecko/20
    <user_agent probability="20">Mozilla/5.0 (Windows; U; Windows NT 5.2; fr-FR; rv:1.7.8)
   </option>
</options>
<sessions>
  <session name="http-example" probability="100" type="ts_http">
  <request> <http url="/solr/player/select?q=playername:yousaf" method="GET" version="1.1"/
  <thinktime value="1" random="false"/>
  </session>
</sessions>
</tsung>
```

The result on running on 1 Riak node are:

Results of 1 Riak Node:

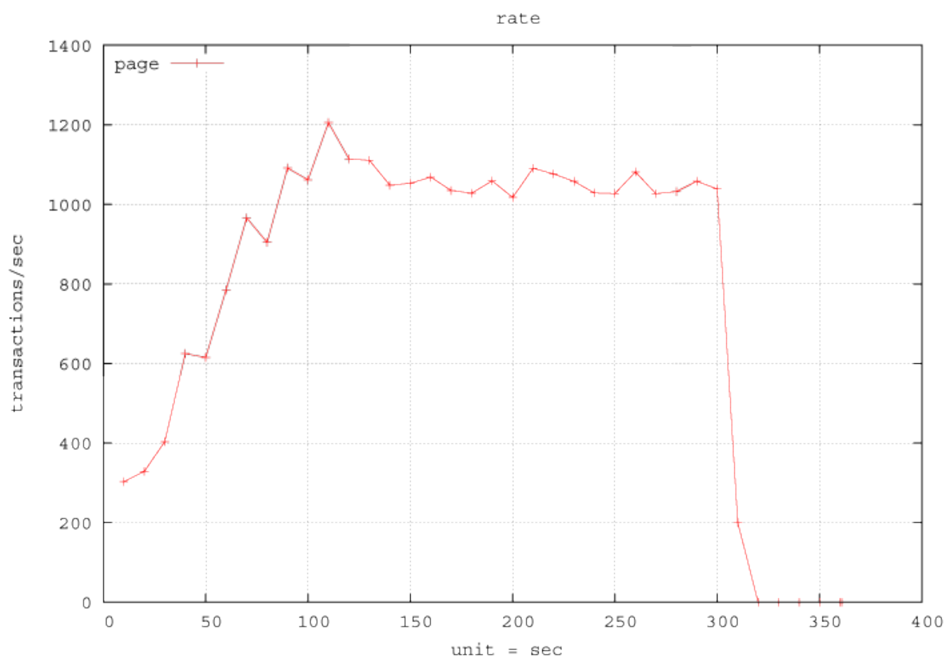

Figure 18: Riak Transaction Rate at 5500 users on 1 Riak node Cluster

Figure18 shows that on 1 node Riak cluster, 1200 transactions per seconds can be performed.

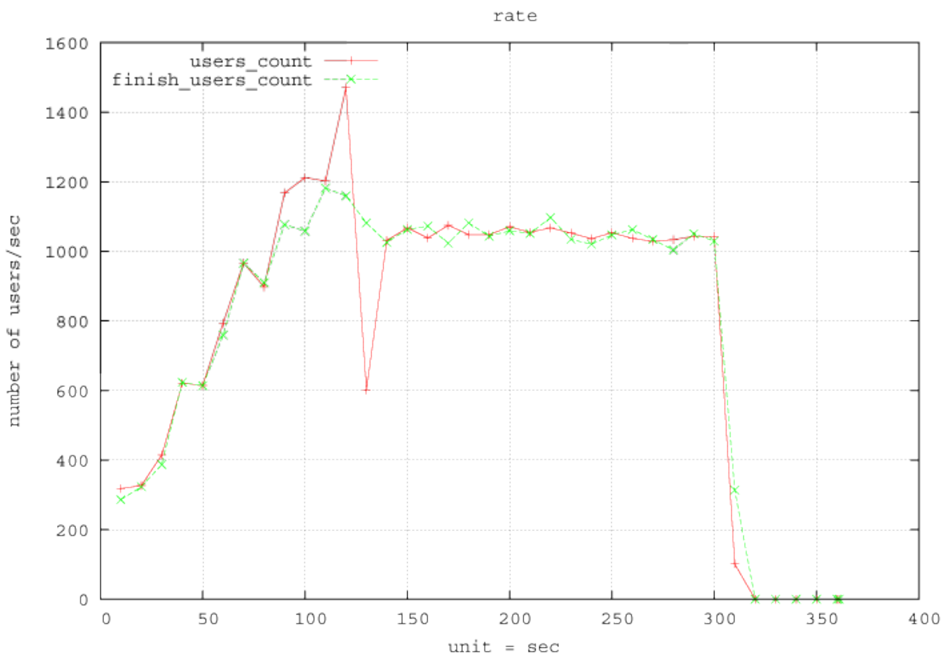

Figure 19: Riak Users Arrival Rate at 5500 user on 1 Riak node Cluster

figure19 shows that around 15,00 users requested to connect to the database per seconds but only maximum of 1200 users per seconds are able to connect to database. Lets see what happen if we add another node in to the cluster.

Results of Adding another node into the Riak Cluster:

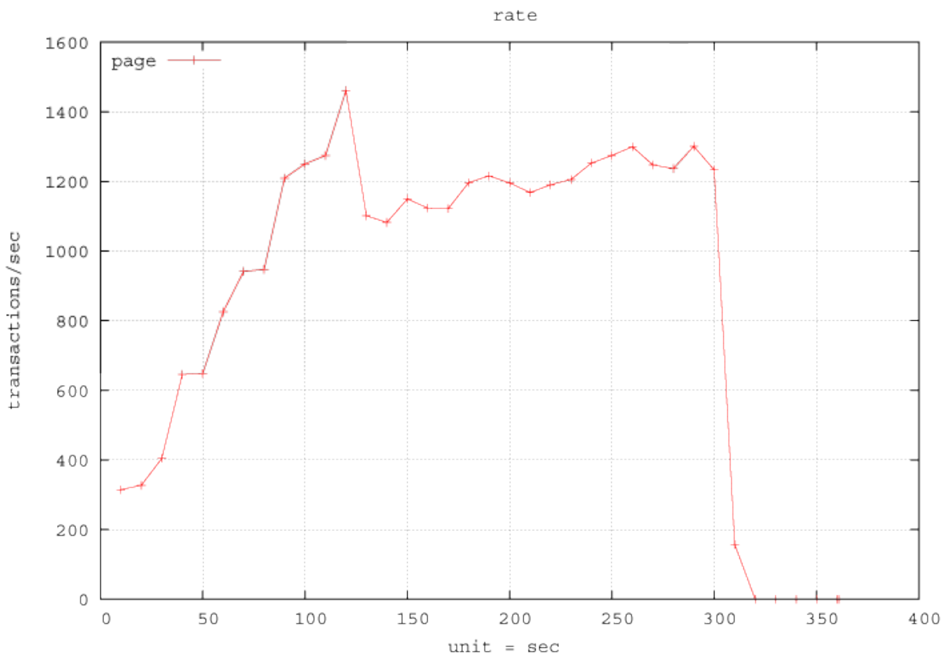

Figure 20: Riak Transaction Rate at 5500 user on 2 Riak node Cluster

By adding another node into the Riak cluster transaction rate per seconds increases from 12,00 to approx 15,00 transaction per seconds as shown in figure 20.

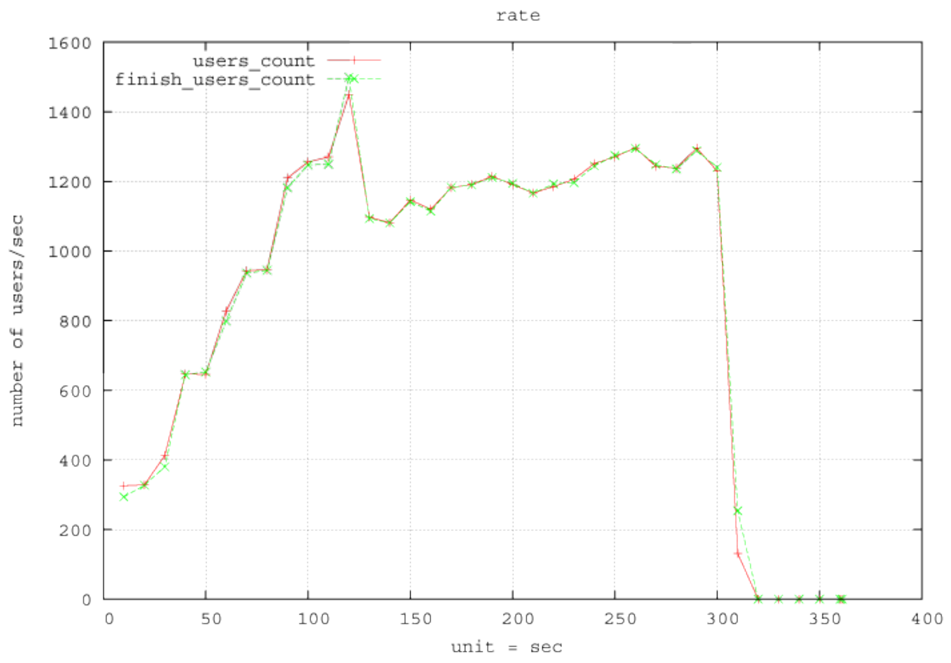

Figure 21: Riak Users Arrival Rate at 5500 user on 2 Riak node Cluster

As in 1 node, Riak could not responds to all the users request per seconds but by adding another physical node in to the cluster scale the database by responding to all the users requests as shown in the figure 21.

6.2.3 Riak Benchmark Evaluation:

Riak benchmark is done using the Basho´s measurement software that determine the number of transactions per seconds executed per second. The benchmark requires a configuration file, which contains necessary parameter to start the benchmark. It executes the given number of worker that simultaneously perform the given task.

```
{mode, max}.
{duration, 5}.
{concurrent, 1000}.
{driver, basho_bench_driver_riakc_pb}.
{code_paths, ["/home/yousaf/basho_bench/deps/stats",
              "/home/yousaf/basho_bench/deps/riakc",
              "/home/yousaf/basho_bench/deps/protobuffs"]}.
```

```
{key_generator, {int_to_bin, {uniform_int, 10000}}}.
{value_generator, {fixed_bin, 10000}}.
{riakc_pb_ips, [{130,238,11,182},{130,238,11,167}]}.
{riakc_pb_replies, 1}.
{operations, [{get, 4}, {put, 4}]}.
```

{mode,max} parameter wants information about the number of operation to be perform on the database. where max means as many operation per seconds possible.[BeB]

{duration,5} parameter need information about the duration of total time for the benchmark.

{concurrent,1000} This parameter shows, how many parallel processes/worker should perform the task.

{riakc_pb_ips} wants the information about the IP's of nodes connected to the Riak cluster.

{riakc_pb_replies,1} wants information about how many nodes should respond to consider a successful read or write.

{operations,[{get,4},{put,4}]} want how many read/get and write/put per worker should perform on the database.

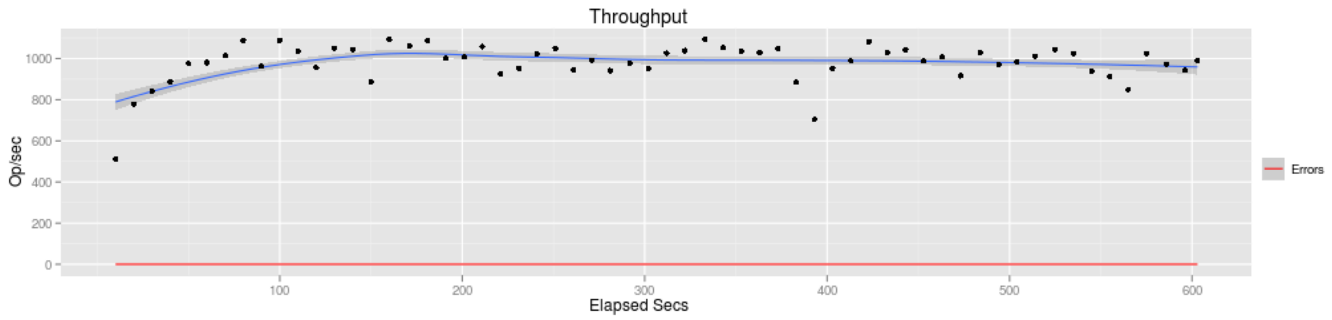

Figure 22: Riak Benchmark

According to figure22, Riak cluster benchmark performs around 1000 operations per seconds in the duration of 600 sec.

7 Conclusion

During the project, I have fulfil the requirements of PikkoTekk to provide a powerful, reliable and scalable solution to store massive multiplayer online game data. Which involves design and implementation of distributed database. To embed the solution into their systems architecture is designed and implemented. According to my research, I found Riak efficient in performance, highly and easily scalable. Its built in load balancer and replication among the nodes in the cluster is very powerful functionality that differentiate Riak among others NoSQL databases.

Unlike any other NoSQL database Riak is purely a distributed database. If your application is massively increasing the no of users and you need to scale your database. If you choose Riak, you only need to add physical node into the cluster according to the growing need of the users. Performance, load balancing and data distribution Riak handles by himself. In comparison with CouchDB, I found CouchDB ease of use but very difficult to scale. Both Riak and CouchDB provides features to increase the performance of reading and writing to the database.

Riak is developed in Erlang, thus inherit concurrent process execution, efficient error handling and stable performance.

8 References

[SORa] Oracle http://searchoracle.techtarget.com/definition/Oracle

[CBT] CouchDB B–Tree http://guide.couchdb.org/draft/btree.html

[DBM] DBMS http://en.wikipedia.org/wiki/Database_management_system

[PKT] PekkoTekk AB http://www.pikkotekk.com

[BeB] Benchmarking with Basho Bench
http://wiki.basho.com/Benchmarking–with–Basho–Bench.html

[RVN] Riak Comparison http://wiki.basho.com/Riak–Comparisons.html.

[TCR] Tunable CAP Controls in Riak. http://wiki.basho.com/Tunable–CAP–Controls
–in–Riak.html

[TSU] Tsung an open–source multi–protocol distributed load testing tool
http://tsung.erlang–projects.org/

[TPI] Mauricio Marin, Carlos Gomez–Pantoja, Senen Gonzalez,Veronica Gil–Costa
Scheduling Intersection Queries in Term Partitioned Inverted Files

[U3d] About Unity http://unity3d.com/unity/.

[Ubas] Unity Basic An Introduction http://technology.blurst.com/unity–basics–overview/

[RSea] Riak Search http://wiki.basho.com/Riak–Search.html.

[NeoTrans] Transactions in Neo4j http://wiki.neo4j.org/content/Transactions.

[VSNO] Visual Guide to NoSQL systems http://blog.nahurst.com/visual–guide–to–nosql
–systems.

[Bre00] Brewer, Eric A.: Towards robust distributed systems (abstract). PODC, 7, 2000.

[CThe1] Brewer Conjuecture and the Feasibility of Consistent, Available, Partition–
Tolerant web services.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.20.1495&rep=rep1&type=pdf

[CThe2] CAP theorem http://devblog.streamy.com/2009/08/24/cap–theorem/

[CThe3] CAP Theorem http://guide.couchdb.org/draft/consistency.html

[CThe4] CAP Theorem http://www.royans.net/arch/brewers–cap–theorem–on–distributed
–systems/.

[CDis] Sharding in CouchDB http://wiki.apache.org/couchdb/Configuring_distributed
_systems

[BCou] The power of B–Tree http://guide.couchdb.org/draft/btree.html.

[Bas] Introduction to Riak http://wiki.basho.com/An–Introduction–to–Riak.html.

[LND] List of NoSQL databases and their families. http://nosql−database.org/

[MRD] MySQL replication Documentation. http://dev.mysql.com/doc/refman/5.5/en/replication.html

[Crep] Couch DB − The Definitive Guide Replication. http://guide.couchdb.org/draft/replication.html

[OMS] Overview of MySQL Server http://dev.mysql.com/doc/refman/5.5/en/what−is−mysql.h

[Cinfo] CouchDB Introduction
http://couchdb.apache.org/docs/intro.html

[HAC] High Avalibility and Scalibility of MySQL. http://dev.mysql.com/doc/refman/5.5/
ha−overview.html

[GPGD Group PostGRESQL Global Development :Replication, Clustering, and Connection Po
http://wiki.postgresql.org/index.php?title=Replication%2C_Clustering%2C_and_Connection_Pooling&oldid=10880