Data Transformation and Query Analysis of Elasticsearch and CouchDB Document Oriented Databases

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

Submitted By
Sheffi Gupta
(Roll No. 801431027)

Under the supervision of:
Dr. Rinkle Rani
Associate Professor, CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

July 2016

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*Data Transformation and Query Analysis of Elasticsearch and CouchDB Document Oriented Databases*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Rinkle Rani* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Sheffi Gupta
(Sheffi Gupta)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Rinkle 15/06/16
(Dr Rinkle Rani)
Associate Professor,
Computer Science and
Engineering Department

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. S. Bhatia)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

The successful completion of any task would be incomplete without acknowledging the people who made it possible and whose constant guidance and encouragement secured the success.

First of all, I wish to acknowledge the benevolence of omnipotent God who gave me strength and courage to overcome all obstacles and showed me the silver lining in the dark clouds. With the profound sense of gratitude and heartiest regard, I express my sincere feelings of indebtedness to my guide **Dr. Rinkle Rani,** Associate Professor, Computer Science and Engineering Department, Thapar University, Patiala, for her positive attitude, excellent guidance, constant encouragement, keen interest, invaluable cooperation. Generous attitude and above all her blessings. She has been a source of inspiration for me.

I am grateful to **Dr. Deepak Garg**, Head of Department and **Dr. Rupali Bhardwaj**, P.G. Coordinator, Computer Science and Engineering Department, Thapar University for the motivation and inspiration for the completion of this thesis.

I will be failing in my duty if I do not express my gratitude to **Dr. S. S. Bhatia**, Senior Professor and Dean of Academics Affairs in the University, for making provisions of infrastructure such as library facilities, computer labs equipped with internet facility, immensely useful for the learners to equip themselves with latest in the field.

Last but not the least I would like to express my heartfelt thanks to my parents and my friends who with their thoughts provoking views, veracity and whole hearted cooperation helped me in doing this thesis.

Sheffi Gupta

Sheffi Gupta
(801431027)

Abstract

With the advent of large complex datasets, NoSQL databases have gained immense popularity for their efficiency to handle such datasets in comparison to Relational databases. The use of web and mobile applications has abruptly increased leading to the generation of mixed data types which are handled by NoSQL databases. These databases has been classified into 4 categories namely Key-Value Databases, Document-oriented Databases, Columnar NoSQL databases and Graph Databases. A number of NoSQL data stores belonging to these classes have been designed for e.g. MongoDB, Apache CouchDB, HBase, Elasticsearch etc. Operations in these data stores are executed quickly. In this thesis, we focus on two most popular NoSQL databases: Elasticsearch and Apache CouchDB. This thesis aims to transfer the heavy dataset from Relational database to NoSQL databases namely CouchDB and Elasticsearch and analyze the performance of these two NoSQL databases on the image data sets as well as text datasets. This analysis is based on the results carried out by transferring bulk data, instantiate, read, update and delete operations on both document-oriented data stores and thus showing how CouchDB is more efficient than Elasticsearch during insertion, updation and deletion operations but during selection operation Elasticsearch performs much better than CouchDB.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Technological advancement has greatly influenced the world of computing today. The introduction of several computing resources and their increasing online application has given rise to massive amount of data, leading to generation of structured and unstructured data known as Big data. Big data is a buzzword used for describing data which is difficult to analyze because of its huge size. As the name indicates, Big data may come across as an association to the volume of data but majorly it is associated with a technology; a technology which helps in handling huge amount of data and its storage with tools and processes. Today, the leading problem revolves around analyzing and extracting data with great speed and high performance. A technique to solve this problem is distributed systems. Distributed systems allows a network to connect with a group of autonomous computers, that shares resources and coordinates their activities for providing a computing facility and able to identify itself as a single unit. A distributed database can be considered as an example of a distributed system in which the storage devices are not connected to a single machine instead(rather) they are managed by a distributed database management system.

The Apache CouchDB database is one such system which follows high scalability and availability without compromising performance. It is used for mission-critical data as it has horizontal scalability and proven fault-tolerance on cloud infrastructure [1]. CouchDB has been used in various websites and Facebook applications like payroll, BBC, Upload booth. With the storage of 10PB of data, its largest production deployments are in CMS (Compact Muon Solenoid Experiment) at CERN [2].

Elasticsearch supports best replication across multiple datacenters with lower latency. Various companies like Netflix, Linkedin, Accenture relies highly on Elasticsearch for storing, indexing and searching the data due to its features like good extension model, plugins, automatic sharding and replication. Netflix alone has deployed more than 15 clusters comprising of 800 nodes [3].

1.1 Rise of Big data

Big data embraces the fact that there is an enormous amount of information around us than ever before that can be put to remarkable new uses. Big data is not same as internet despite the fact that the internet helps in collecting and sharing huge amount of data easily, instead Big data revolves around the fact of driving enough knowledge from the data when dealing with large body of information than what we can learn with smaller amounts.

During the third century BC, the entire human knowledge was believed to exist at the Library of Alexandria. But considering today's situation, with current data present in the world; if every living person is given sufficient amount of information, then it will be 320 times more as predicted to be in Alexandria library- estimated to be 1200 Exabyte [4]. And if all this data is written on CDs, then it will take five stacked up piles of CDs that all would reach to the moon.

This rise of data wasn't there till 2000, a quarter of information of the world was digital back then and the rest of the information was recorded on paper and Analog Medias. For every three years, the digital data is said to expand with a speed of doubling itself leading to the need to handle this data. With this acceleration in digital data, now the trend has changed leading to downfall of nondigital information which is not even 2% of the stored information.

The enormous size of data sometimes misled to be just in terms of size. Rather of just focusing on quantification of data, nowadays more importance is being given to transform that data into new form of value. This is called datafication. For example, initially location was associated with longitude and latitude, but now with GPS satellites. Even relationships and "likes" on social media sites are being datafied, putting the data into brilliant use now. Now a computer is not taught how to do things, such as translate a language or drive a car but is now fed with enough data to check with the probability of substituting a more appropriate word in case of translation and also if the traffic light is green/red in case of driving. This is made possible with the help of powerful processors and smart algorithm which uses math based on statistics using smart data models called NoSQL databases [5].

1.2 Challenges with traditional databases

Some of the challenges faced by traditional databases are discussed as follows:

a. Traditional databases are not capable of handling data with mixed data types like text, images, audios, videos etc.

b.Scalability of large volumes of data cannot be achieved through traditional databases. For example, the AADAR project of Indian Government has 15-20 petabytes of data that cannot be scaled through traditional databases.

c. RDBMS databases are not capable to handle rapid change of data.

d. RDBMS are considered to be a mismatch for developing Modern Applications.

e. It is quite difficult to handle such large variety of data with RDBMS model which includes features like locking, parsing, logging etc.

f. The much focused property to be followed by RDBMS model is ACID property restricting the scaling requirements of today's world, which is much relaxed in NoSQL databases.

1.3 NoSQL Databases

In the database community, the term NoSQL is identified as "Not Only SQL". It is considered to be the store of various unstructured databases including key-value databases, column family databases and document databases [6]. The need of NoSQL arises from the incompatibility of relational databases to handle gigantic volume of data over internet and to cope with new trending technologies like cloud computing, big data etc. The very basic requirements of cloud computing technology are dynamic scalability and handling large data easily, but relational databases require more complex and costly hardware to fulfill these requirements because they are designed to work on single server itself to handle its ACID properties in an efficient way.

As Scalability can be attained in two ways – horizontal scalability and vertical scalability. Horizontal scalability means scaling out. To achieve horizontal scalability, commodity servers are added to the existing node. On the other hand, vertical scalability is known as scale up. This is mainly delivered by traditional databases by increasing resources for a

single machine itself. With increase in data, vertical scalability is considered to be an inappropriate option and in some cases, it is not even a feasible choice. So, here comes the need to shift to non-Relational databases that delivers horizontal scalability. Recently evolved non-Relational Databases also called as NoSQL databases use horizontal scalability. Several NoSQL databases have been designed so far like MongoDB [7], Oracle NoSQL[8], Cassandra[9], HBase[10] etc.

CAP theorem is the base of NoSQL databases. It states three basic requirements that should be taken care of while designing various applications in a distributed system. These basic requirements includes Consistency, Availability and Partition Tolerance. The first requirement ensures consistency of the data after performing an operation on it. For example, after the execution of an update operation, all the clients should see the same data at same time. Availability guarantees service to work with no downtime. Partition Tolerance means fault tolerance, that is, if the communication failure occurs between servers, then also the system should continue to function. Practically, when working in distributed system it is not possible to acquire all the three requirements. As shown in Figure 1.1, NOSQL databases works by combining any of these two requirements. For example, CouchDB fulfills A and P.

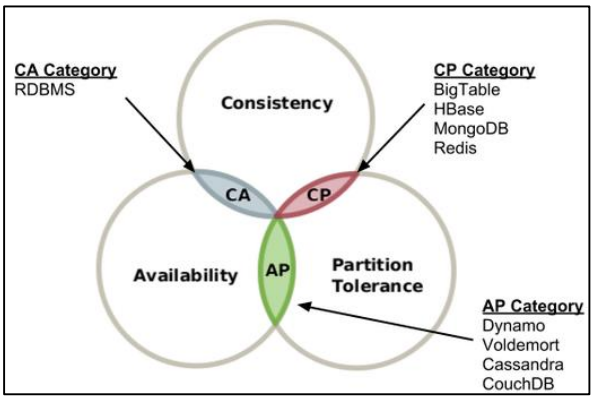


Figure 1.1 CAP Theorem [11]

These databases are designed by taking on the risk of not following the most important ACID properties of traditional databases. Instead, they support their own property called BASE (Basically Available, Soft state, Eventual consistency) in order to fulfill the need of scaling out the increasing traffic over Web and to make possible to run on large clusters which the traditional databases were unable to do. Basically Available guarantees

availability in terms of CAP theorem. Soft State indicates change in the state of system even at the times when input is not changed. This happens due to changes going on because of Eventual Consistency. In contrast to relational databases, all NoSQL databases are schema-less. As implicit schema accesses the data in NoSQL databases, so here migration needs to be performed carefully [12].

1.3.1 NoSQL Classes

NoSQL data models are grouped into four classes capable of incorporating more than one model at once are discussed as follows:

1. **Key-Value Databases:** Key-Value store is defined as a distributed persistent associative array consisting of a key which is a unique identifier for a value. This model is considered to be the fastest way for retrieving data by a known key, with the help of simple GET, POST, PUT and DELETE commands. It can be used to share data between application instances like to store user session data or distributed cache. For example, Amazon Simple DB, Redis, etc.
2. **Document-Oriented Databases:** These document-oriented data models are designed to store semi-structured data. In contrast to relational databases, these document-oriented databases can have any number of fields in a particular document and these can be changed as per the requirements without affecting other documents. The most common format used to represent objects is JSON. For example, CouchDB, Elasticsearch, MongoDB, etc.
3. **Columnar NoSQL Databases:** These data stores are also known as wide table data stores handles sparse data in an excellent way. These are designed for organizing data on the basis of individual columns. Each column family can be identified with the help of key indexes. Here, a single family is a collection of columns. These store data in a way that columns are contiguously stored rather than rows, leading to better performance when from large number of columns in a result set only few of them are required to retrieve. HBase and Google's Big Table are main inspirations for column-oriented databases.
4. **Graph Databases**: The Graph Data model has been designed for building a graph structure by linking a lot of different objects with each other. Here, objects are

considered as nodes in a graph and links are represented through the edges. Various social network websites like Twitter and Facebook produce several terabytes of data highly interconnected with each other. So, to handle such large interconnected data, Graph databases are extensively used. Graph databases are highly scalable, flexible and map directly to object-oriented programming models. These data stores are considered to be the backbone of social networking sites. For example, InfoGrid, Infinite Graph, Neo4j, etc.

1.4 Migration from relational databases to NoSQL

Currently, all major developing organizations whose data is abruptly increasing are now shifting from relational data models to NoSQL data models such as Apache CouchDB and Elasticsearch. Big Data has majorly derived the data analytic technology like Elasticsearch. Effectively scaling data storage has actually given opportunity to analyze the Big Data. In this global database market, only 1/4th of data models deal with analytics and rest are operational databases. Here, we are comparing Apache CouchDB and Elasticsearch to analyze which of the two has better performance in regard of CRUD operations. CRUD operations include Create, Read, Update, and Delete. These data models are considered to be the persistent storage of data.

1.5 Couch DB

CouchDB is a NoSQL database designed to modularize, scale up and focuses on completely embrace the increasing demands of revolutionary world. The name of Apache CouchDB has been derived from the idea of "easy to use", that's why a phrase is attached to CouchDB "Its's time to relax". CouchDB runs across a cluster of servers in a distributed way. It is a cross platform document oriented database which is available on various operating systems created by Damien Katz in 2005 and now, maintained by apache Software Foundation. It is presented in key-value maps using JavaScript Oriented Notation (JSON) [13].

The breakthrough of mobile applications and dynamic web applications, the data has increased abruptly and so the requirement of efficient data models has become important. CouchDB is one such platform that handles a variety of structured, unstructured, image,

audio, email data. The data stored in any format can be easily queried to retrieve relevant data. At a time, CouchDB has the capability to connect with large number of databases which further consists of large number of documents in JSON format, that is, it uses JSON to store documents oriented data. There is no need to define structure and constraints on data, which was the major drawback of relational database.

Futon is a built-in web application through which administration of CouchDB is supported.

The advantages of using documents are:

- Documents defines a construct in a single entity which would have required several tables in a relational database to be presented properly.
- No need for implementing expensive joins in documents due to implementation of embedded documents and arrays.
- Dynamic schema leads to fluent polymorphism.

1.5.1 Key Features

a) **Document-level ACID semantics:** CouchDB has the capacity to handle large number of concurrent readers/writers by the implementation of multi-version concurrency control without any conflict [14].

b) **Replication and Built for offline:** CouchDB can replicate with by taking care of off-line mode, that is, it stores multiple copies of the data and synchronizes the data when the device comes back to online mode

c) **Eventual Consistency:** For providing both partition tolerance and availability from CAP theorem, CouchDB guarantees Eventual Consistency

d) **Map/Reduce Views and Indexes:** In CouchDB, views are constructed through a JavaScript function triggering the Map/Reduce operation.

e) **HTTP API:** Items in CouchDB can be accessed by the unique URI via HTTP by using HTTP methods for performing basic operations.

f) **Fault tolerance: M**ulti Master Replication leading to effective fault tolerant architecture.

g) **Concurrency:** Excellent concurrency features because of the fact that it has been written in a Erlang programming language

1.5.2 CouchDB Architecture

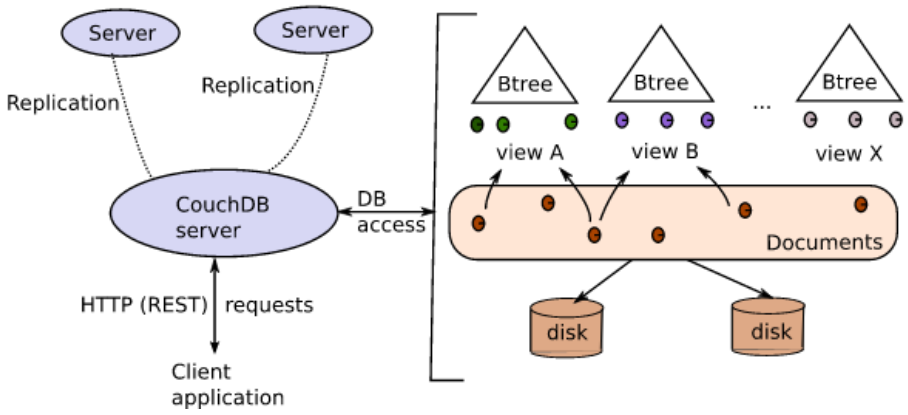


Figure 1.2 CouchDB Architecture [15]

- As shown in Figure 1.2, an effectual B-Tree storage engine is the heart of Apache CouchDB. B-tree allows to perform operations of search, insertion, deletion and updation in logarithmic time. Fig shows that B-Tree storage engine is used by CouchDB for all documents, internal data and views. There exists two predefined functions in CouchDB: map function and reduce function, collectively called MapReduce. For query language, it uses JavaScript with MapReduce. MapReduce is used by CouchDB for computing results of view. These functions lends the view computation to be in parallel and incremental way and are the real reason for success of CouchDB in handling variations in unstructured data, independently indexing each document store and processing them in parallel by generating key-value pairs corresponding to each document . View results from the combination of map reduce function: list of key/value pairs.
- As shown in Figure 1.2, CouchDB inserts views into B-Tree storage, sorted by key. In CouchDB, view and document results are accessed by key [16].
- B-Trees streams the rows according to key range and helps in fast searching of rows through these keys and thus, resulting in huge performance gains. The data can be partitioned on multiple nodes, with no effect on querying the data.

- CouchDB server is the central module handling user requests, view engine, drives the storage engine and replicates for evaluating queries and retrieving final results.

Three CouchDB strategies are followed by storage engine for managing data storage: Multi-Version Concurrency Control (MVCC), Append-Only Database and Streaming Data.

1. Multi-Version Concurrency Control (No Locking)

In a relational database, a table is considered as a single data structure. During the updation of a row in a table by a particular user, the system needs to confirm that no one else has the authorization to either read or update that row until it is being updated. As shown in Figure 1.3, the relational database system handles this through the use of "lock". So, if multiple clients are accessing a table, they needs to wait in a queue if any one of them is trying to update that data. This serial execution of requests wastes rich amount of server's processing power by spending more time in finding who is allowed which authorization and in which order, rather than performing its actual request.

But in CouchDB, instead of locks, MVCC is used here for managing concurrent access to databases.

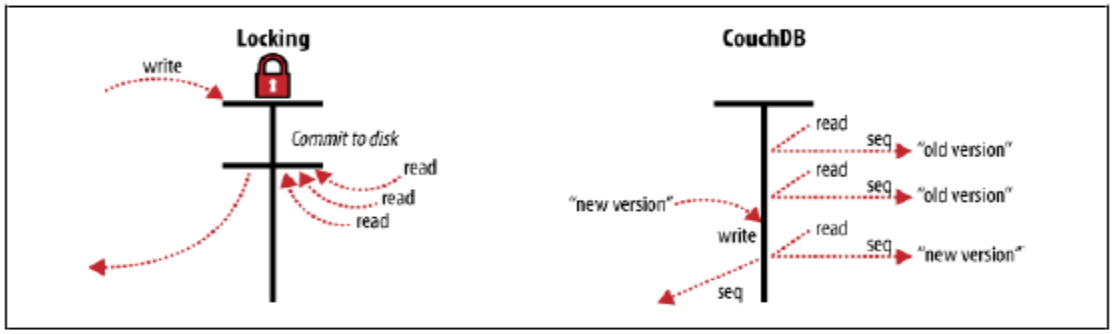


Figure 1.3 Difference between traditional locking and MVCC mechanisms.

So, by providing the ability to run parallel requests, MVCC ensures that even during high load, CouchDB can run to full speed every time.

2. Append-only database

Incremental B-Tree is considered as the core of CouchDB. CouchDB provides versioning to the documents. If a particular document gets updated, then a new version gets assigned to it and it gets saved over the old one. So, at this point the application has 2 versions of a

particular document: old and new, that is, files in CouchDB are never overwritten in storage engine. Instead, a new transformed document after updation or deletion gets appended at the end of file present in storage engine. This strategy orders the documents on the basis of time. As operations are not overwritten, database backup is much easier with low cost.

3. Streaming storage

The raw storage of CouchDB is a stream of data consisting of a header and segments corresponding to documents. The header of a raw storage which is a doubly-linked list contains database's meta-information. Each segment is defined as a structure containing data, header and footer. Header stores length of data and location of former segment, while footer stores location of next segment.

Each modification to the data is stored as a segment by just appending new version segment at end of existing database file, showing that former version is now out of date. Delete operation shows that from now onwards, all former versions won't be accessible to any user (by just appending a small segment).

1.5.3 Design Aims

The need of designing CouchDB has emerged from the data storage requirements of web developers that can be accessed with both JavaScript and HTTP-Requests web technologies. There is no requirement of schema creation and so, no database design errors are possible. The design aims of CouchDB are focused on the following needs:

a) **Scalability (time):** For getting adequate response times, CouchDB has been designed to extend a web application easily by simply adding cluster nodes. This is required because of quick and unanticipated growth of websites with time
b) **Scalability (space):** The quick growth of popular websites also require to handle increasing user content by more disc space [17]. CouchDB aims to handle this situation also
c) **Adequate to web data:** as web data is unstructured, so CouchDB handles it through schema-less database accounts

d) **Convenient for web developer:** CouchDB has been designed on the policy of easy to learn and use for any web programmer

e) **Availability:** CouchDB takes care of requirement of service uptime in globally distributed sites.

1.5.4 CouchDB Data Model

a) Data as Documents

CouchDB stores data as documents in JSON format .Each of the document can have varying number of fields means to say that there is no need to define the structure of each document as was required in relational databases during creation of tables. So, CouchDB is considered to be a schema-less database. Every document in CouchDB contains metadata like their unique ID and _rev number.

With each updation to a document, a new version of the existing document is created with different revision id and is appended to the history tree of that document. So, it handles the documents history in a way similar to Web-based or wiki document-management.

CouchDB documents tend to have all data for a given record in a single document, whereas in a relational database information for a given record is usually spread across many tables.

As a result of the document model, data in CouchDB is more localized, which dramatically reduces the need to JOIN separate tables. The result is dramatically higher performance and scalability across commodity hardware as a single read to the database can retrieve the entire document.

b) CouchDB Dynamic Schema

CouchDB documents can vary in structure. For example, all documents that describe users might contain the user id and the last date they logged into the system, but only some of these documents might contain the user's identity for one or more third-party applications.

Fields can vary from document to document; there is no need to declare the structure of documents to the system – documents are self-describing. If a new field needs to be

added to a document then the field can be created without affecting all other documents in the system, without updating a central system catalog, and without taking the system offline. In documents, fields are neither bound to predefine their data types nor to any size limit, instead, they can store numerous types of data like numbers, datetime, text strings, Boolean, etc. Each field in a document is required to have a unique name.

1.5.5 CouchDB Eventual Consistency

Talking about the CAP theorem, CouchDB has been designed to be partition tolerant and highly available with its eventual consistency feature. These features in CouchDB keeps the data safe. Here, Eventual consistency is achieved from verification and replication techniques. It also depends on the CouchDB mode in which it is working [18]. If CouchDB works with single server, then it is strongly in consistent state, but if it is in master-master replication, then Availability and Partition Tolerance (AP) are followed.

1.5.6 CouchDB Data Management

CouchDB stores large amount of semi-structured documents. An implicit schema is followed by each CouchDB object by having an identifier serving as a unique key. These are very much different from SQL databases, which are designed for storing querying interrelated and highly structured data. Due to the handling of document-oriented data, CouchDB has been used in the development document-oriented Web applications to a great extent.

CouchDB interacts with client application through the implementation of REST interface. The server answers with messages encrypted in JSON through HTTP. A GET request is sent to the server and in return, we receive a JSON-encoded message:

$ curl –X GET http://mycouch.org
{“couchdb”:”Welcome”,”version”:”1.0.1”}

The server has the ability to maintain several number of JSON documents.

CouchDB storage file consists of “contiguous” regions to store documents. Here, two B-Tree indexes are present for speeding up access to documents.

1. By_id_index (document id is used as a key). It lookups the document through document id, by pointing to list of revisions from the last compaction it also stores the revision history.
2. By_revision_index(monotonically increasing number is used as a key) during each updation of a document, a revision gets generated. It is helpful in keeping track of last replication synchronization point and last view index update point.

a) Map/Reduce View

In CouchDB, querying data is done through views. Views are mainly of two kinds:

1. **Temporary views:** These views doesn't get stored in database and are accessible through HTTP POST request to URI*/{dbname}/_temp_view*
2. **Permanent views:** These views get stored in design documents. For executing permanent views, HTTP GET request is made to URI*/{dbname}/{docid}/{viewname}*

The definition of a view can be described in 2 functions: Map and reduce. The map is responsible for generating contents of view, thus, it is not optional and is a JavaScript function. User writes these map functions, which iteratively checks the documents in the database, to find those that match the condition that the user has specified in the map function [19]. After the collection of results from the function, CouchDB inserts these results into view's B-Tree index using emit() function, which is a built-in function generating single row information from key and value of the map function. This B-Tree now consists of list of key-value pairs.

The tuple in index which is built for storing the view is identified by the key. And the emitted value is stored at the B-Tree leaf nodes.

```
function(doc) {
   if(doc.age && doc.age > 15 && doc.name)
      emit(doc.name,doc.age);
}
```

In this example, all documents having key age value greater than 15 are emitted [20].
Unlike Map, Reduce is an optional part. Reduce makes computation on the documents stored in B-Tree non-leaf nodes of that index. It takes the list of values and reduces it to single value [21].

b) Self-contained data CouchDB stores the data as a self-contained document which helps in mimicking the real world documents like business cards, tickets and bills helping the various users to easily understand the semantics of data.

c) Storage efficiency with compaction As we have discussed that overwriting of any data is not allowed in CouchDB, that is older and deleted versions of objects are never removed from CouchDB's raw storage, leading to very high percentage of invalid documents. So, to overcome this situation, Compaction is provided in CouchDB [22]. And after compaction of older versions, the only documents having latest versions will be shown to the user.

d) Pluggable storage architecture for application flexibility

CouchDB embraces two key trends in modern IT:

a. **New apps.** Organizations are expanding the range of applications they deliver to support their business
b. **Technology rationalization**. CIOs are rationalizing their technology portfolios to a strategic set of vendors they can leverage to more efficiently support their business.

With the use of pluggable architecture, CouchDB is able to address diverse application demands reducing the complexity for developers and various organizations.

e) ACID Semantics: As CouchDB implements MVCC feature, so it becomes easy for the CouchDB to handle large number of concurrent reads and writes to a database without any conflict, leading to ACID semantics.

1.5.7 Replication and Fault Tolerance

The biggest strength of CouchDB is to put multiple copies of same database in synchronization. Replication involves 2 databases: source and destination that can be either on same or other CouchDB instances. The purpose of replication is to replicate all the changes made to active documents in source database to destination database, that may be either updation of a document or deletion of the document. For this, *changes Feeds* in source are compared to destination documents. If they are found to be different, changes are immediately transferred to destination in batches. This task also has the ability to establish checkpoint documents on destination, in order to ensure the

synchronization of restarted task, for example, after its crash. If the sending node initiates replication task, it is known as push replication, but if receiving node initiates it, then it is known as pull replication.

CouchDB consists of a _replicator database consisting of multiple documents, each describing a particular replication process. Replication status can be known from active tasks API.

a) Fault tolerance: As the underlying technology used for implementation of CouchDB is Erlang/OTP, which is the real backbone for CouchDB to act as highly scalable and fault-tolerant database [23].

b) Conflict resolution between replicated instances and Revision Management Optimistic concurrency paradigm is followed by CouchDB's update model. Revision id is associated with each CouchDB document. During each update, this revision id gets changed. During synchronization of documents between source and destination databases, only the documents with fresh revision id gets updated to target database. If the conflict occurs between two databases, a special attribute will be flagged with the affected document "_conflicts": true. And in result, it will show the version with latest revision id and other revisions will be stored as older revisions. So, CouchDB always ensures a consistent data set. A revision tree is present on both sides containing documents history by storing list of older revisions.

If the document is updated in both databases, then full synchronization can be attained by a "pull replication" followed by a "push replication" or vice versa. If one tries to update a document's older version, then CouchDB gives the 409 conflict for the rejection of update operation.

1.6 Elasticsearch

Elasticsearch is a real time distributed social analytics engine mainly designed to organize the data in order to make it easily accessible. It is built as an open source on top of Apache Lucene [24] which is a full text search engine.

It is a distributed document store, it stores all objects as JSON documents. These documents are indexed by default and are schema free, so that we don't have to define fields for data types before adding data [25]. It handles the fault tolerance by redundantly

coping the data, and maintaining the high availability of data. It also provides the feature of multitenancy for querying multiple indices independently. Communication with Elasticsearch is done through HTTP REST API.

1.6.1 Key features:

a) **Real-time data: A**ctive polling in Elasticsearch refreshes the data of screen with live data.

b) **Advanced Search and query increasing performance: S**earching indices in Elasticsearch are easy due to flexible query features. This is because of the fact that it has been built on top of Lucene (full-searched information retrieval library) which is written in Java. It can search multi-languages, geolocations and so on.

c) **Configurable and Extensible:** Elasticsearch configurations are changeable even when it is running. It do possess several Extension plugins in order to enhance its capability.

d) **Provides REST based interface**: Elasticsearch is easily accessible through the RESTful based API using JSON over HTTP, that is, Elasticsearch is API driven. Responses given by Elasticsearch are in JSON format, which further adds up to its easily readability. It is a schema free database.

e) **Per-operation persistence:** Elasticsearch keeps track of all the transaction log changes made on multiple nodes [26]. Thus, reducing data loss chances to great extent.

f) **Multitenancy:** Multiple indices can be stored in single cluster or server, each having multiple "types". A single query can be sufficient to search data of multiple indexes.

g) **Highly available:** Replicas of shards corresponding to each index can be created and changed dynamically from 0-many, providing high availability of data.

h) **Horizontal Scaling:** Elasticsearch has the ability of infinitely scale up the system and can even work with petabytes of data.

i) **Distributed Storage:** Due to creation of multiple shards of a single document on different clusters each configuring an index, Elasticsearch is said to be a distributed engine.

j) **Scalability and Flexibility:** As discussed, plugins can be attached to Elasticsearch providing high scalability and flexibility to the application. As shown in Figure 1.4, the number of replicas can be adjusted or changed dynamically in Elasticsearch, but the same is not possible for shards as they are fixed for each index. So by taking future server expansion into consideration, the number of shards must be allocated properly during the first time itself.

```
$ curl -XPUT http://localhost:9200/log-2012-12-27/ -d '{
    "settings": {
        "number_of_shards": 10,
        "number_of_replicas": 1
    }
}'
```

Figure 1.4 Dynamic Configuration Change Query

1.6.2 Elasticsearch Architecture

When Elasticsearch node is started, it multicasts to find other nodes present in the same cluster and gets connected to them.

The process of Elasticsearch used by an application has been illustrated as follows:

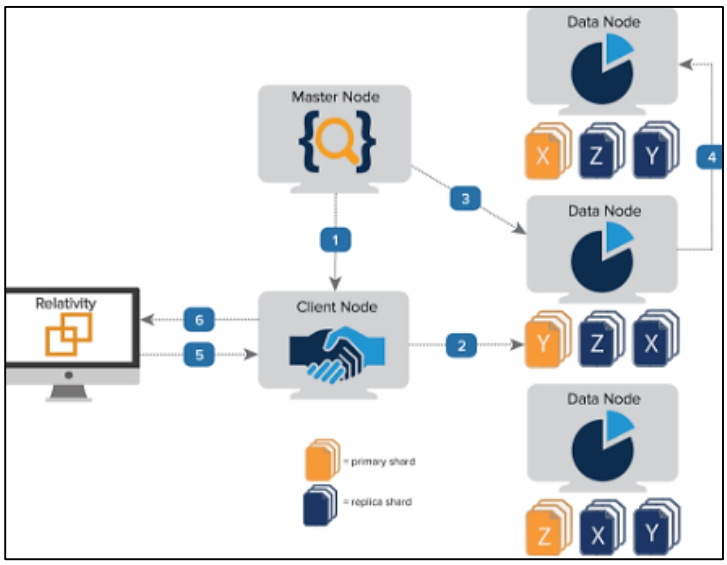


Figure 1.5 Elasticsearch NoSQL Architecture [27]

Indices: Indices are used to store data in Elasticsearch. Data is stored in the form of documents and is read from the indices. Apache Lucene library is used for reading and writing data from an index.to build a single index in Elasticsearch, one or more Apache

Lucene indices may be used with shards and replicas. An index in Elasticsearch is created and it gets split into one or more shards which resides on different nodes.

Mapping: A type in an index is defined by mappings.

Node: Node is a running instance of Elasticsearch server. As shown in Figure 1.5, when a node is started, it searches for a cluster to join with it and during large data storage, multi-node Elasticsearch cluster is used due to inability to get fit in a single server [28]. All nodes have information about all other nodes in a cluster and can send client requests to any node. There are different types of nodes each designed for a specific purpose and are discussed as follows:

a) **Master node:** Each cluster is connected to a single master node which is chosen automatically. This master node controls the cluster.
b) **Data node:** this node is set to true by default. These hold data and carry out data related operations like search, CRUD, aggregations, and so on.
c) **Client node:** this is set to false and is connected to both master node and data node. It is a "smart router" load balancers, used for forwarding cluster-level requests and data-related requests to master node and appropriate data nodes respectively using transport module.
d) **Tribe node:** this is the special case for client node connecting multiple clusters and performing various operations across connected clusters.

Cluster: Cluster is a group of nodes. The concept of clustering helps to store large volumes of information, which was not possible with a single server. A master node handles Primary shard which is the first place where the document is stored when it is indexed. After indexing the document in primary shard, Replicas of the primary shard will get copy as well. The spreading of data to distinct physical Lucene indices is done to achieve clustering. These Lucene indices are collectively known as shards and this process is known as sharding. This is done automatically by the cluster and happens during the index creation itself.

Replica Shard: This is just a copy of primary shard. So, it provides a back-up plan if primary shard goes down and also, replica shards has the ability to enhance the

performance of Elasticsearch as these replicas helps when load increases and it becomes impossible for a single node to handle entire requests. The number of replicas can be adjusted manually by the user, by adding and removing them any time.

Gateway: During the time of work, Elasticsearch iterates for collecting the information regarding indices settings, cluster state, etc. this data is stored in the gateway.

1.6.3 Design aims

a) **Peer-to-peer architecture**

For the prevention of failures, automatic replication of data is necessary providing not even single point of failure (SPOF) easing mutual monitoring and data interchange [29].

b) **Easily scalable**

It should be scalable easily not only in terms of capacity, but also in terms of data

c) **Unrestricted data organization**

Users in Elasticsearch can adjust to the already defined data model because it doesn't put any restrictions for organizing data in index.

d) **Near Real Time(NRT)**

Versioning and searching in Elasticsearch should be near to real time. This is achieved from the distributive nature of Elasticsearch.

1.6.4 Internal Data Storage Management

Elasticsearch manages its data by following 2 mechanisms: Analytics and Inverted indexes [30]

1) **Analysis:** Analyzers are the combination of three logical parts:
   - **CharFilter:** this component makes modification of the underlying char stream directly and is an optional logical part like removing HTML tags
   - **Tokenizer:** Extraction of multiple terms from a particular text string
   - **TokenFilters:** Modification, addition or deletion of tokens are carried by this component.

During the creation of an index for a document, a default analyzer and mapping gets attached to it. The document gets converted into stream of tokens. After the generation of

tokens, the output gets filtered by Token Filters. This entire process is known as Analysis. User can also create their own analyzers.

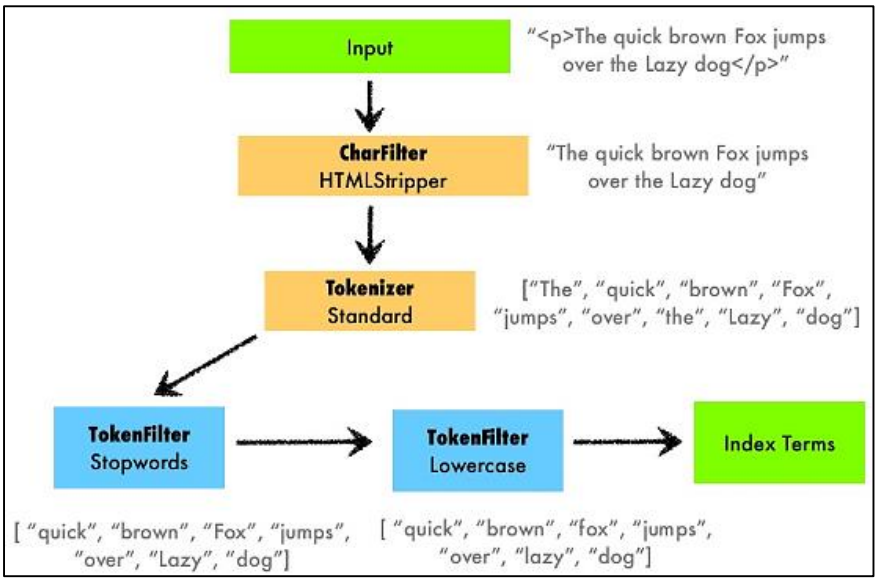


Figure 1.6 Analysis Process

In the Figure 1.6, the analysis process of html embedded document is shown.

2) **Inverted indexes:** Inverted indexes are the data structure storage for Elasticsearch, whereas, the data structure used for an index in SQL servers is binary tree. As shown in Figure 1.7, the tokens created above in the analysis process gets stored in an internal structure known as inverted index, mapping every unique term to document. So, due to inverted index, Elasticsearch is able to do text analytics and faster data search. A term is associated with various attributes like term position, term count, and so on.

   After the mapping of tokens, documents are stored on disk. Along with analyzed document, original documents can also be stored. The original documents are stored in system fields "_source". Inverted index structure depends on which analyzer has been chosen for indexing.

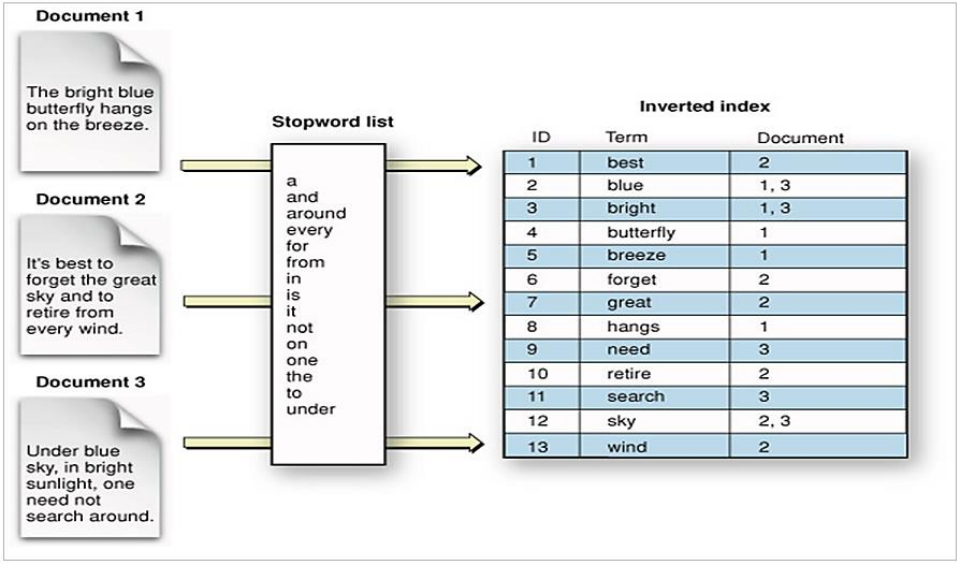


Figure 1.7 Inverted Indexing [31]

1.6.5 Data Models

The data that is indexed or searched in Elasticsearch is a document. These documents do have following properties:

Self-contained: A document consists of various fields with their corresponding values.

Hierarchal: documents in Elasticsearch can have nested structure also, that is, a document can also contain another document in it along with fields.

Flexible structure: there is no need to define structure of documents.

Indexing a Document

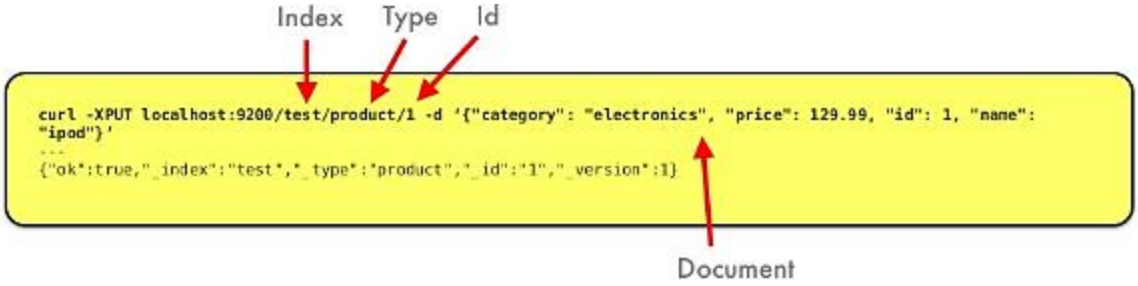


Figure 1.8 Document Indexing

During the creation of a document, all the fields in the document are indexed. Here, as shown in Figure 1.8, test is the name of the index that maps the product with id=1. There is no need to define structure of document to be indexed and the data type of fields are based on the interpretation made by the Elasticsearch from the values of fields during

indexing of first document. A JSON object is returned as a response acknowledgment by Elasticsearch. Here, Id is an optional part. If the user doesn't specify document's id, then Elasticsearch auto-generates it. Id is returned by the Elasticsearch in predefined **id** field for the JSON response.

Along with this, there is one more predefined field in JSON response that is **_version**, where a version number is kept for every indexed document. A document in Elasticsearch can be reindexed or updated with varying attributes and its version will get incremented automatically. The resemblance of Elasticsearch to the Relational Database is as shown in Table 1.1.

Manual index creation we can also map the fields manually during the creation of index as during automatic mapping, mapping is done "on the fly" by introspecting the documents to be indexed.

Table 1.1 Elasticsearch similarity towards SQL database

MySQL	Elastic Search
Database	Index
Table	Type
Row	Document
Column	Field
Schema	Mapping
Index	Everything is indexed
SQL	Query DSL
SELECT * FROM table …	GET http://…
UPDATE table SET …	PUT http://…

1.6.6 Security

In Elasticsearch, shield provides the main features of security. It does so by providing four main features [32]:

a) **Role-Based Access Control**

It gives permissions to the users on the basis of their roles, that is, it sets index and granular cluster level permissions corresponding to every user of Elasticsearch.

b) Authentication System Support

Integration of LDAP-based authentication systems and Active Directory provides additional features to the users.

c) Encrypted Communications

The protection of data from intruders is done through node-to-node encryption. Shield is able to provide protection to the data travelling on the wire through HTTP secure client communications and SSL/TLS encryption.

d) Audit Logging

Shield makes the logs of activities done in the Elasticsearch deployments by recording unauthorized attempts made to access the data or number of login failures occurred.in short, it tracks all the attempts made to access the Elasticsearch.

e) Failure detection

During regular processing, master node keep an eye on all the available nodes, if they are working or not. If any of the nodes are not available for specified amount of time, then that node is considered to be as broken and the process of handling that failure node starts. Now, the shards on that failure node are gone, and other nodes have to take responsibility for each of the lost shard. This is known as rebalancing of cluster. If the lost shard is primary shard, then a new primary shard is elected from remaining replica shards.

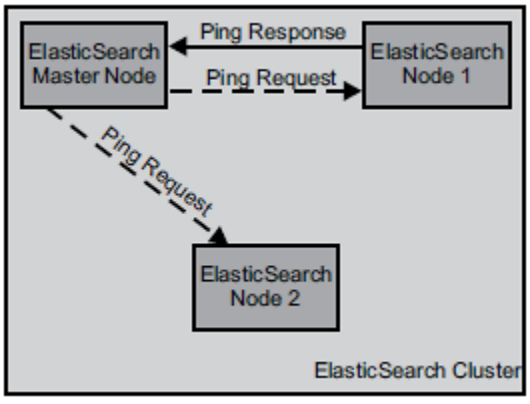


Figure 1.9 Failure Detection

As shown in Figure 1.9, it consists of three nodes cluster; two data nodes and a single master node. The master node send ping requests to data nodes and waits for their response and if the response doesn't comes, that node gets removed from the cluster.

1.7 Structure of the Thesis

The rest of the thesis is organized in the following order:

- **Chapter 2 Literature Review -** It surveys the history of big data, problems faced by traditional databases. Further, it discusses the history of CouchDB and Lucene indexing from where the Elasticsearch has emerged.
- **Chapter 3 Problem Statement -** It discusses the problem statement and corresponding research gaps. Further, the objectives of problem statement and the methodology used for solving it have been discussed.
- **Chapter 4 Implementation Details -** The first few pages of this chapter have been devoted to explain the steps taken for the installation and implementation of both data models and discusses the steps followed for generating the huge data set while the rest of pages contains implementation snapshots explaining in detail the proposed work.
- **Chapter 5 Results -** This chapter includes analysis of the results. Further, it also compares the work through various tables and graphs.
- **Chapter 6 Conclusion and Future Scope -** This chapter concludes the thesis, the contribution it makes and the future scope.

Finally references, list of publications and video presentation link have been appended.

Chapter 2 Literature Review

The emergence of big data along with the demand of data-intensive computing has given rise to many NoSQL databases. Two of them are Apache CouchDB and Elasticsearch. In this section, the review of various methods suggested by researchers for handling these increasing demands of NoSQL databases and to enhance their performance is done along with the review of meeting the various challenges and barriers.

2.1 Evolution of Big Data

C. L. P. Chen and C. Zhang [33] performed analysis on big data and concluded that many sources like business processes, tractions and social networking sites has produced big data that is both structured and unstructured. Various business application being developed are commonly comes under the category of web-oriented large scale and data intensive applications. These applications can then be accessed through various devices including mobile phones. Big data sizes varies greatly from dozens of terabyte to various petabytes all in a single set of data. It is quite difficult to capture, manage and process such a huge data.

Y. Huang and T. Luo [34] has defined big data as a combination of many datasets like structured, semi-structured, unstructured heterogeneous and homogeneous data. They have presented a Nice Model committing transfers to very less demand periods having a lot of idle bandwidth that is available under this model. This bandwidth can be restructured for transmission of bigger amount of data without actually affecting other users inside the system. Store and forward approach is being used by the Nice model through the utilization of staging servers. The variations in the bandwidth and differences in the time zones are accommodated by the model. In order to solve problems like security, compression and routing algorithms, suggestions have been made by them for implementation of newer algorithms.

W. Fan and A. Bifet. [35] proposed that these days real time streaming data analysis is becoming the most efficient and the quickest way for obtaining useful knowledge

regarding the system and in case of detection problem, it allows organizations to respond quickly in order to increase the level of performance. The data that is built in large chunks every day is known as "Big data". Tools like Apache CouchDB, Elasticsearch, Cascading, Scribe, Apache mahout, etc. are extensively used for mining of big data. Thus, he supported the fact that our capability of handling several Exabyte's of data is primarily based on the presence of rich amount of techniques, datasets and software frameworks.

B. Purcell [36] stated that Big Data comprises of large chunks of datasets which the traditional database systems cannot handle. So, technique is required for storage of big data consisting of structured, semi-structured and unstructured data and that technique is known as object based and multiple clustered Network Attached Storage (NAS). This is made simple through the use of MapReduce that helps in locating and selecting data which satisfies the query. They have also discussed various challenges that the big data has posed on current businesses.

E. Bertino, D. Jewell, M. Hammersley and Y. Li [37] stated that indexing is very efficient in database, especially during big data size. It is considered as a challenge due to various reasons. As the generated index must be smaller comparative to the velocity and volume of Big Data that are larger of the order of magnitude and along with this index should be faster also for searching the content from billions or trillions of data sets in seconds.

2.2 Limitations of traditional databases

S. V. Phaneendra and E. M. Reddy [38] presented how RDBMS tools are facing problems while handling huge data known as big data. They have elaborated the fact that how big data varies from regular data mainly in terms of five dimensions that are volume, velocity, value, variety and complexity along with describing the various challenges faced during handling of big data like data privacy and search analysis.

S. Agarwal *right* [12] has presented BlinkDB, which is a parallel query engine running interactive SQL queries on large data sets. BlinkDB was developed by considering two ideas:

1. A framework that should be adaptive and should work on stratified samples from original data, that are multi-dimensional in nature maintaining and building samples that are being collected over time.
2. A sample selection strategy that is dynamic in nature and helps in selection of proper sized samples in accordance to the accuracy requirements and query response time.

J. Lee, S. H. Sang and M. J. Lee [39] stated that object-oriented databases do not provide any database support for improving timeliness or guarantees for deterministic behavior of an real-time application. Neither they provide any feature for supporting concurrency control of real-time applications. Considering these major drawbacks of object-oriented data models, it is highly recommended to start with a simple with which it is possible to extent to process the real-time applications.

C. Tyagi and S. Batra [40] proves that graph databases have the better performance for objective tests that is they retrieve data much faster than relational databases. And these are much more flexible for defining new relationships in the existing schema which is a relatively tough in relational databases. The comparison has been made using queries of both the databases and concluded that Neo4j, database used for graphs is much more useful for commercial purposes like social networking and website link structure.

With the discovery of cloud based services [41], most of the organizations started relying on cloud based services as these are based on pay as-per need rule. These services are also becoming popular as they support scalability and availability of data, which is the basic requirement of today's world. As these requirements of the cloud based services are not supported by the Relational Databases, instead these are supported by NoSQL databases. Thus, there comes the great need to shift from Relational Databases to Cloud (NoSQL) Databases. This is achieved through Data Migration methodology. So, as the choices of data storage among NoSQL databases has been increasing at a rapid rate, there also grows the need to migrate data between these available databases and to evaluate in which situation the existing databases perform better.

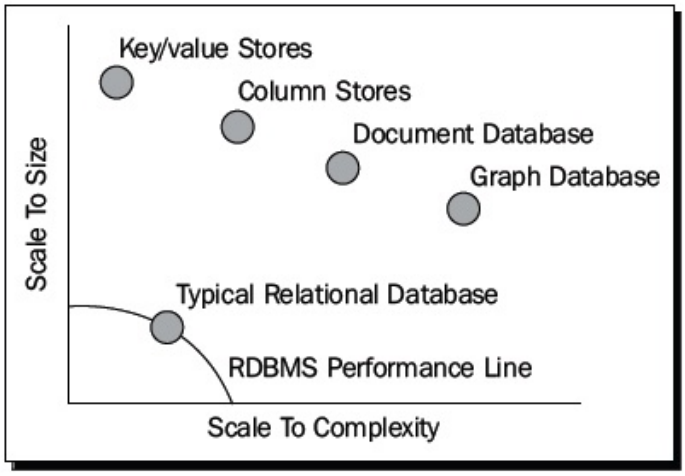


Figure 2.1 Scalability of Relational Databases and NoSQL Databases

Figure 2.1 clearly represents the performance difference between Relational Databases and NoSQL databases. This is due to the fact that Relational databases are not designed to work in a distributed environment, so relational databases are neither in tune with cloud based services nor with storing and handling Big Data [42].

3.3 Selection criteria of Databases

When designing a new application, selection of the database must be in accordance to the application's requirement. The different criteria's which should be considered for selection of databases are as follows:

- **Development criteria:** This criteria includes the drivers that are availability, data consistency, query functionality and data model. These factors affects functionality of an application along with the time taken for building it.
- **Operational criteria**: It includes scalability, performance, high availability, security, management, backups and data center awareness. During the lifetime of an application, the operational costs affect the project's TCO (Total Cost of Ownership) to a great extent. Hence, these costs adhere to SLAs decreasing administrative overheads.
- **Commercial criteria**: This criterion includes support, pricing and licensing. The database that is chosen should be aligned with the business.

2.3 Elasticsearch

Elasticsearch is NoSQL database created by Shay Banon in 2010. But before this, a Compass is being created by Shay Banon in 2004. But during the release of latest version of Compass, he realized the need for creating a scalable search solution capable of storing and searching a large amount of data at lightning speed using JSON over HTTP, leading to the release of Elasticsearch in February 2010. It was all designed by Shay Banon due to the need of distributing the data and to easily scale out the data with accurate results in fraction of seconds.

The Butler Group, [43] in a report "Enterprise Search and Retrieval" stated that almost 10% corporate salary costs are taken away due to difficulties faced by employees for finding required information.

Smartlogic [44], in "Mind the Enterprise Search Gap" stated that less than half of enterprise storage and analysis applications meet the expectations of their users. And the same results are shown by the reports carried out by IDC in 2007 named "Quantifying Enterprise Search". So, considering the difficulties carried out by the results of reports, Elasticsearch came into existence.

Q. Liping and W. Lidong [45] studied the performance comparison between Oracle and Lucene indexing and concluded that Lucene is more superior from oracle in terms of full text searching of huge dataset. And stated that keyword searching doesn't demand CPU task. Building of indexes is in favor of both large memory and CPU, but not for disk I/O. For reaching higher performance, it can utilize aggregated I/O bandwidth.

Y. Zhang and J. Li [46] stated that the retrieval of indexing in Lucene is extremely fast and it can also be adjusted to further improve it. He proposed a new retrieval sorting Lucene algorithm by adjusting Lucene indexes which performs much faster than Lucene and Page rank algorithm. He states that Lucene can be much more useful for full-text searching than any other algorithm.

A. B. Mathew, P. Pattnaik and D. M. Kumar [47] proposed two new approaches for indexing in Hadoop along with the implementation of Lucene in Hadoop and concluded

that Lucene performs much faster than any other full-text search approach. He concluded it, by performing comparison of Lucene, Lindex and Hindex on the basis of CPU utilization and number of bytes read by each of them in specific amount of time in Hadoop environment.

2.4 CouchDB, MapReduce and Indexing

The breakthrough in data-intensive computing and Business intelligence has led to the rise of distributed computing clusters. Making such clusters using relational databases was very costly and complicated to handle, so to make such clusters inexpensive and highly utilized, software frameworks and application programming model is required.
For easy writing of scalable parallel applications which can handle and exploit huge clusters for data-intensive computation, MapReduce programming model is being implemented. MapReduce is designed in such a way that it provides scalability, parallel computation and allows every node to handle its respective slice of dataset while loosely coordinating with other nodes. The programming model helps in improving the performance of application as the increase in number of computer nodes increases the parallelism which can be exploited. This programming model of MapReduce is implemented in various new-generation NoSQL databases, that includes Hadoop, CouchDB and so on.

CouchDB uses Erlang/OTP programming language for its implementation. The most popular applications of open source Erlang include Ejabberd system, CouchDB, MochiWeb library, RabbitMQ. Due to the use of Erlang programming language in CouchDB data model, CouchDB is able to provide scalability feature across multicore and multi-server clusters.

K. Lee, Y. Lee, H. Choi, Y. D. Chung and B. Moon [48] performed a survey on MapReduce assisting several database organizations to have better understanding of various technical aspects of MapReduce framework by describing its various features and characterizing them. Author also mentions the challenges and open problems faced by MapReduce framework.

Ranger, 2007 stated that unlike relational database models, CouchDB doesn't infuse indexing power directly. In fact, CouchDB uses index that is further assisted through the use of map and reduce functions of predefined MapReduce framework. These functions encourage parallelism and absorbs any variations present in document structure of data and allows the indexes for performing independent computations. Map functions creates indexes capable of limiting the window of targeted data and for seeking optimal efficiency, Reduce queries are permitted to run.

H.M.L. Dharmasiri and M.D.J.S. Goonetillake [49] have successfully managed to produce and implement a federated NoSQL system with three different types of NoSQL solutions. By implementing a federated system between MongoDB, CouchDB and Cassandra, this task has been achieved. They proved by the various types of tests and evaluations it can be finally exclaimed that the NoSQL federation is a feasible solution and it will actually take off a lot of time and effort taken in large scale data migrations. Since these choice options have a very large impact on the final-end performance of NoSQL systems, careful selection must be done.

R. Cattell, [50] stated that versioning is also an important feature of CouchDB. Multi Versioning Concurrency Control (MVCC) in CouchDB helps to provide exclusive locking functionality on data files during the ongoing write operation. Version conflicts get resolved in CouchDB with the merging of conflicted data into one file.

CouchDB has been used as an efficient document-oriented database in many advanced applications for example Z. Liu, Y. Wang and R. Lin [51] has used CouchDB for storing and analyzing its applications log data from a Platform as a Service (PaaS). This is due to the fact that CouchDB uses B-Trees as its underlying structure leading to its lower cost and faster speed to store the data. In the same way, L. Ramaswamy, V. Lawson and S. V. Gogineni [52] has used CouchDB for storing sensor network data. This became possible with CouchDB due to its feature of holding the meta-data (data about data) along with data from the feed. And allowing to query any attribute present in the data storage.

D. Havlik, M. Egly, H. Huber, P. Kutschera, M. Falgenhauer and M. Cizek [53] stated that CouchDB can be used in cases when there is no need to explore off-line replication

capabilities. In CouchDB, multiple instances of a single database can exist that can be queried and updated independently without interrupting each other and allowing synchronization only when the instances are communicating with each other. This feature of CouchDB is used extensively in applications where clients and servers are not always on-line and are also in need to provide lower latency and access of local data to remote clients.

Chapter 3 Research Problem

3.1 Problem Statement

In the current scenario, data has been growing at an exponential rate. This growing data requires to be stored in databases that can support scalability and can store large amount of data efficiently. Due to Business Intelligence, these days the value of Big data has also increased to such an extent that storing and managing data in a well-built way has become the necessity of today's market. This all happens to make the data ready for analysis purposes and build large scale applications that will be helpful to users throughout the world. To handle all this, many non-relational databases have come into existence in today's market.

The recognition to these databases is increasing due to their support in making applications taking care of specific needs of users, requiring real-time changes within click of hands, ability to store semi-structured data, scalable nature, etc. Due to the changing needs, organizations make their data models to shift from relational to NoSQL databases, and its their responsibility to analyze which of these NoSQL databases best fit to their requirements.

Thus, there comes the need for migration algorithms and to analyze which NoSQL database is best suited for any particular purposes. This need has emerged from the problem with relational databases unable to handle unstructured data, images, videos, audios, and inability to perform data analytics operation, which NoSQL databases can do very easily.

Thus, an analytical and performance comparison of CouchDB and Elasticsearch using real-time query processing implementation assists in understanding the various efficiencies of such databases. The comparison done on CRUD principle (Create, Read, Update and Delete) is basic to any form of dataset and application system. So, these queries provide us results with real data and helps in making decision for choosing the perfect database for an application.

3.2 Research Gap

1. Transformation of data from Relational to NoSQL Document-Oriented Databases. The main challenge here is to map a schema based information to a schema-less data model.
2. Performance comparison between CouchDB and Elasticsearch is needed for assessing the differences in their performance such that an application or business information system is able to determine whether the requirements of their system are being met correctly with an appropriate data set.
3. It is important to use a fast and effective system for handling data these days. A lot of theoretical comparisons has been made among various databases for selection of a better database corresponding to available dataset but there is no focus on practical implementation of these databases.

3.3 Objectives

In the light of above discussed research gaps, following objectives have been formulated.

1. To study various existing NoSQL databases
2. To propose and design an approach for
    a. Transformation of data from relational database into document-oriented NoSQL database without loss of any information and schema-mapping
    b. To insert multimedia data in document-oriented database by following the file stream methodology
    c. To execute real life queries for insertion, updation, selection and deletion without the use of "CRUD" queries
3. To perform the comparative analysis of CouchDB and Elasticsearch based on their relational database transformation, insertion, updation, selection and deletion operations.

3.4 Research Methodology

Data migration from relational databases and comparative analysis of Elasticsearch and CouchDB are the main focus here and it is implemented by using following methodology:

1. Embedding documents methodology is used to store Relational dataset in Document-oriented NoSQL databases. File streams have been used to insert multimedia data.
2. Java methods have been created managing key/value pair for carrying out insertion, updation, deletion and search operations.
3. Latest version of Elasticsearch and CouchDB are used as NOSQL databases. Query analysis of CouchDB and Elasticsearch is performed on the basis of the execution time in milliseconds corresponding to each operation.

The real work is to decide a cluster for a particular application. The selection requires a solid understanding of size of data along with complexity of creating and handling a multi-node cluster. Thus, this thesis tries to answer the problem by performing practical comparison between the two databases such that appropriate choices can be made by the users while selecting a database. These choices will help in proper handling of the large data sets present.

Chapter 4 Implementation Details

Focus of this thesis is on transforming SQL dataset into NoSQL dataset and on the Performance Comparison of CouchDB and Elasticsearch database systems. For this, all the implementation work has been done in Eclipse IDE using Java framework. For the database systems – Apache CouchDB and Elasticsearch, their respective libraries have been used for testing a certain set of data to evaluate performance between two databases. The evaluation of this dataset is done using tables and graphs on the basis of time taken to perform different set of operations. Finally it has been concluded that CouchDB takes less time than Elasticsearch in insertion, updation and deletion operations, whereas for search operation, Elasticsearch performs more efficiently than CouchDB. The dataset considered for the following was an unstructured data consisting of text and images taken for the reason as NoSQL data models deals better with such type of data.

The implementation starts with installing different sets of libraries used in project that are as follows:-

MYSQL-Version – 5.5

CouchDB: Apache-Version – 1.6.1

Apache Tomcat Server-Version – 7.0.69

Elasticsearch: Version – 2.3.2

Java: Version - 1.7

4.1 CouchDB

4.1.1 Installation Steps for CouchDB

a. First install Java on the computer system.

b. Then download the latest version of Apache CouchDB-1.6.1 from the Apache website and installed on the system.

c. Make appropriate changes in the configuration files of system directory and make directories in accordance to the requirements of use, before actually running CouchDB on the system.

d. After this set the Java_Home in the CouchDB configuration file couch-env.sh.

e. Then the ports which are going to be used in CouchDB are defined in core-site.xml file where CouchDB will be connected.

f. After completing above steps CouchDB can be started by running CouchDB.bat file present inside the bin folder.

g. CouchDB is now running on the system. Its connection status can be verified by running URL *http://127.0.0.1:5984* in browser.

h**.** After CouchDB has been configured and installed successfully, it can be connected with Eclipse IDE in order to proceed with the implementation.

4.2.2 CouchDB Implementation

a. Create a project for CouchDB to proceed with the implementation.

b. Import the marvel plugins and all other necessary jars that are required for the implementation.

c. Set Index and BaseUrl in resource.properties, so that Tomcat URL gets connected to the CouchDB URL through the REST API.

d. Connect Java application with MYSQL databases for traditional datasets.

e. Using the Java API, make connection with CouchDB and create documents in data model necessary of approx. 1GB using multimedia data and by transferring heavy data from MYSQL database. All the data in CouchDB is stored as bytes.

f. Now begin the implementation of certain set of operations using Java on the selected data set, enabling to access the database through Apache Tomcat Server's IP address.

g. We start with the data insertion process into the document-oriented database system and record the time(in milliseconds) taken to complete this operation.

h. After this the database is processed with data selection operation on the same data set and record the results for it as well.

i. The third set of operation to be performed is of updating the data.

j. The fourth set of operation performed on the dataset is of deletion.

k. Repeat these four operations 10 times each and the results are recorded in form of time (milliseconds) and are represented through Tables and Graphs to complete the analysis.

4.2 Elasticsearch

4.2.1 Elasticsearch Installation

a. Configuration of Java variables has already been done during CouchDB installation so there is no need to perform those steps again.

b. The very first thing required to do is to download Elasticsearch and install it on the system.

c. Before starting Elasticsearch update the configuration files and create directories for storing data in the system.

d. Define the variables such as Java_Home in Elasticsearch configuration file.

e. Next create the directories for storing logs and data of Elasticsearch.

f. Now, Elasticsearch is ready to get started. To do so run the Elasticsearch.bat file present inside the bin folder.

g. Elasticsearch is now running on the system. Its connection status can be verified by running URL *http://127.0.0.1:9200* in the browser.

4.2.2 Elasticsearch Implementation

a. Eclipse IDE has already been installed for Java code compilation, so no need to repeat this step again.

b. Create the Java project for Elasticsearch to proceed with the implementation.

c. Further import marvel plugins and all other necessary jars that are required for the process of development.

d. By using the Java API, connection is made with the Elasticsearch database using Index and Types stored in resource.properties.

e. Connect Java application with MYSQL databases for traditional datasets.

f. After making a successful connection with Elasticsearch, create documents in data model necessary of approx. 1GB using multimedia data and by transferring heavy data from MYSQL database similar to the dataset used in CouchDB.

g. After this, perform same four sets of data operations (Insertion, Selection, Updation and Deletion) as done in CouchDB.

h. Repeat these four operations 10 times each and the results are recorded in form of time (milliseconds) and are represented through Tables and Graphs to complete the analysis.

4.3 Migration Roadmap

Migration of Relational data to Document-oriented databases has been done using following steps:

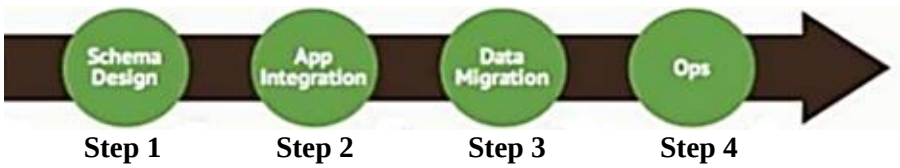


Figure 4.1 Steps taken to migrate data from Relational to NoSQL Databases

1. Schema Design

The big data used in this project has been created manually with the use of GenerateData.com website and this data is then imported into SQL database.

Following is the Relational Database named as User that will be used as an example database. Table 4.1 shows the attributes taken for example Relational Database User. Attributes belonging to primary keys of that relation are underlined in the Table 4.1. A foreign key relation between two tables is represented as $R_i.P \rightarrow R_j.Q$, consisting of a foreign key relation between the attribute P of relation R_i and attribute Q of relation R_j. Relational Database User consists of a set of tuples over three relations. These relations are explained in Table 4.1

Table 4.1 Schema details of Relational Database "user"

Relation Name	Attributes	Primary Key	Foreign Key Relation
User	User, Username, Edu, College, City, Email_id, Phone, Company, Job_location, Experience	User	None
Comments	Comment_id, Comment_text, User	Comment_id	Comments(User) $\xrightarrow{fk}$ User(User)
Follower	User, Page_Id	Page_Id	Follower(User) $\xrightarrow{fk}$ User(User)

2. App Integration

Now, a Java application with document-oriented NoSQL database has been created which makes a connection with Relational Database management system (MYSQL) for importing its database.

3. Data Migration

After establishing connection with Relational database, the methodology for transferring large data into CouchDB and Elasticsearch NoSQL databases has been developed. As, Relational data is not able to hold multimedia data. So, during the transformation of relational data into NoSQL data, the methodology of inserting multimedia dataset has also been implemented along with it. After successful data transfer, the dataset will look as shown in the Figure 4.2.

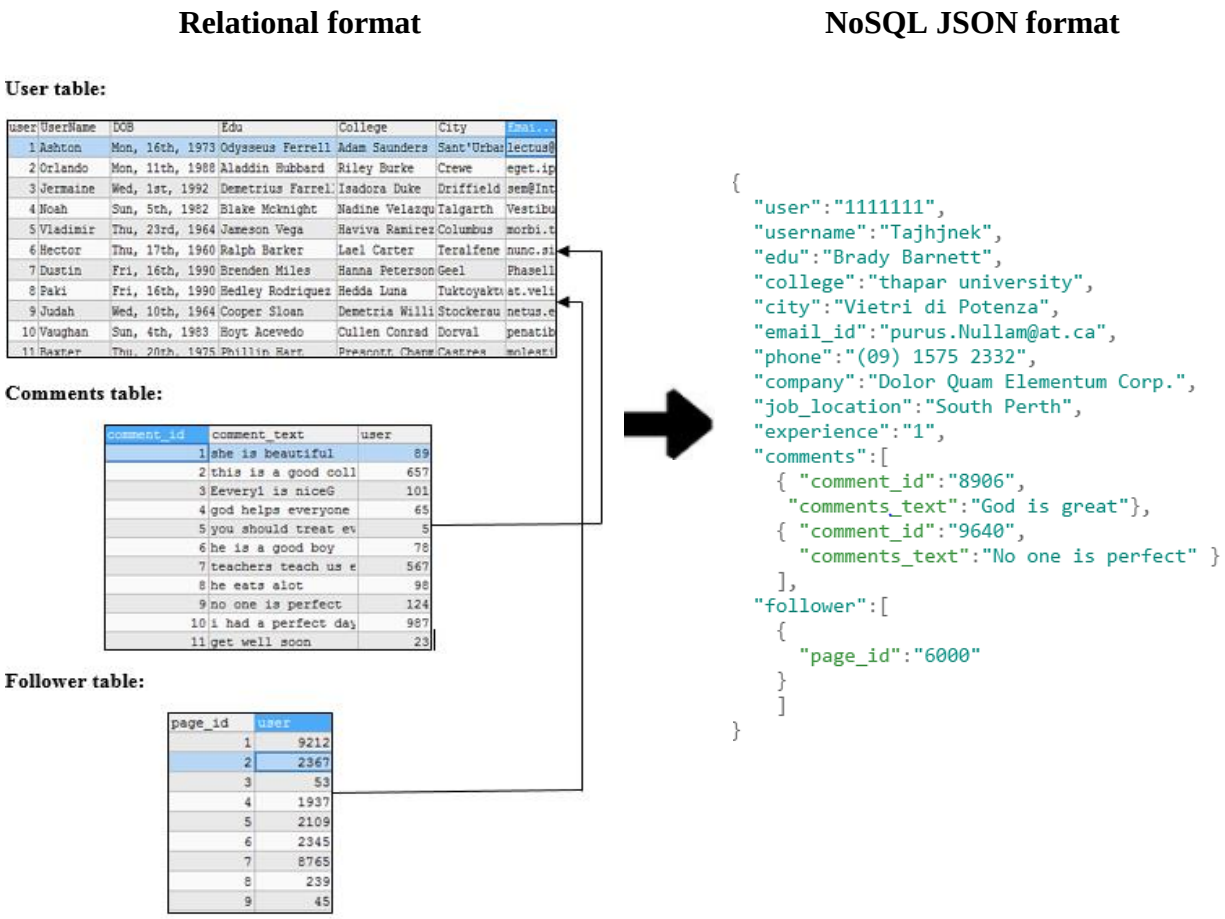


Figure 4.2 Data Models: Relational to Document

4. Operations

After transferring data, required operations are performed on the database.

4.4 Implementation Snapshots

Various screenshots of implementation have been attached along with the detailed steps for better understanding.

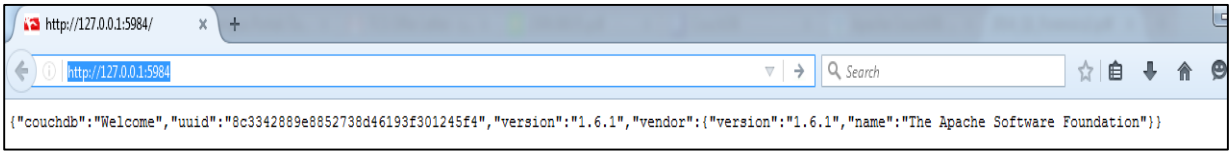


Figure 4.3 CouchDB Running State

As shown in the Figure 4.3, CouchDB is started using Couch.bat file and it is now running on the system. Its connection status is checked by running the URL ***http://127.0.0.1:5984*** in the browser.

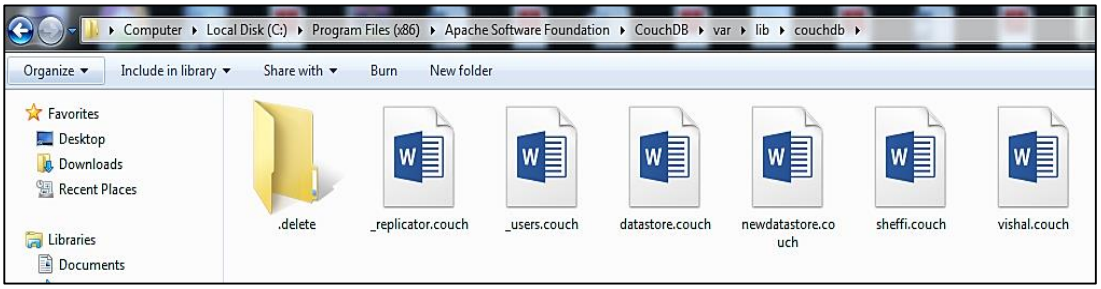


Figure 4.4 CouchDB's Storage Directory

As shown in the Figure 4.4, CouchDB storage directory consists of all the indexes created with respect to each user.

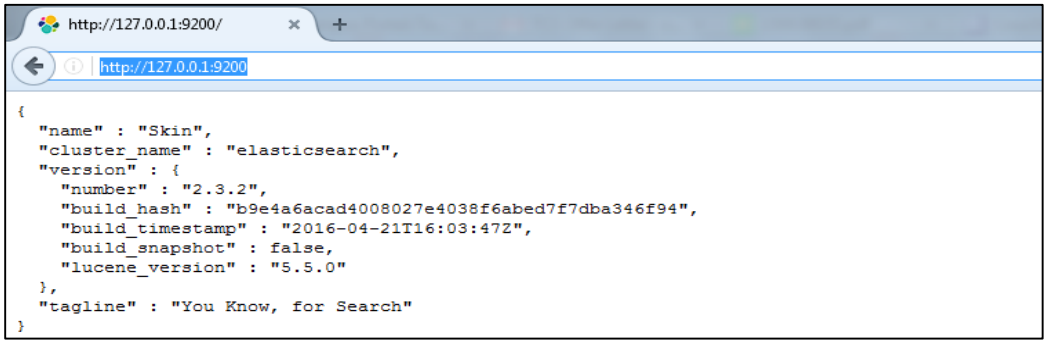


Figure 4.5 Elasticsearch Running State

As shown in the Figure 4.5, Elasticsearch is started using Elasticsearch.bat file and it is now running on the system. Its connection status is checked by running the command ***http://127.0.0.1:9200*** in the browser.

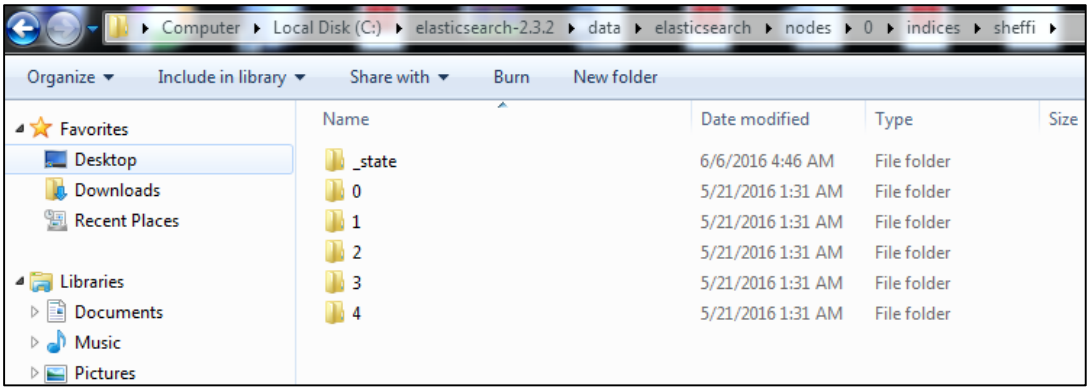


Figure 4.6 Elasticsearch's Storage Directory

As shown in the Figure 4.6, Elasticsearch storage directory consists of all the indexes created with respect to each user type. As the Figure 4.4 shows index is named as "Sheffi", in which 0, 1, 2, 3, 4 are its types.

```
// Inserting a document in Elasticsearch database
    @RequestMapping(value = { "/insertDocument" }, method = {
            org.springframework.web.bind.annotation.RequestMethod.POST }, consumes = { "application/json" })
    @ResponseBody
    public DomainResponse insertDocument(@RequestBody String document) throws Exception {
        String url = ResourcePropertiesUtil.getResourceProperties("BaseUrl");
        String result = makeconnection(url, document, "POST");
    }

// Inserting a document in CouchDB database
        String url = ResourcePropertiesUtil.getResourceProperties("BaseUrl");
        String index = ResourcePropertiesUtil.getResourceProperties("Index");
        HttpClient httpClient = new StdHttpClient.Builder().url(url).build();
        CouchDbInstance dbInstance = new StdCouchDbInstance(httpClient);
        CouchDbConnector db = dbInstance.createConnector(index, true);
        ObjectMapper mapper = new ObjectMapper();
        JsonNode node = mapper.readTree(document);
        Map<String, Object> doc = mapper.convertValue(node, Map.class);
        doc.put("_id", node.get("user").asText());
        db.create(doc);
    }
```

Figure 4.7 Inserting a document in Elasticsearch and CouchDB databases

As shown in the Figure 4.7, both databases take JSON as input document. Elasticsearch performs all operations directly through HTTP Client, but in CouchDB Erktop library have been used, inside which all the used methods are defined and CouchDB performs all operations with the use of this Erktop library HTTP Client. In Elasticsearch, Primary key is generated by Elasticsearch itself, there is no involvement of user, and vice-versa in case of CouchDB. As seen in above figure, POST operation is performed directly on JSON, but in CouchDB, JSON node is mapped to Map object.

```
// Retieving documents from Elasticsearch database
    @RequestMapping(value = { "/readDocument/{username}" }, method = {
            org.springframework.web.bind.annotation.RequestMethod.GET }, consumes = { "application/json" })
    @ResponseBody
    public DomainResponse readDocument(@PathVariable(value = "username") String username) throws Exception {
        String url = ResourcePropertiesUtil.getResourceProperties("BaseUrl") + "/_search";
        List<String> arrayList = new ArrayList<String>();
        arrayList.add("user");
        arrayList.add("college");
        int size = arrayList.size();
        String user = "*" + username + "*";
        String jsonArray = "[";
        for (int i = 0; i < size; i++) {
            jsonArray = jsonArray + "\"" + arrayList.get(i) + "\"";
            if (i < size - 1) {
                jsonArray = jsonArray + ",";
            }
        }
        String json = "{\"query\":{\"query_string\":{\"query\":\"" + user + "\",\"fields\":" + jsonArray + "}}}";
        String result = makeconnection(url, json, "POST");
    }

// Retieving documents from CouchDB database
    {
        String url = ResourcePropertiesUtil.getResourceProperties("BaseUrl");
        String index = ResourcePropertiesUtil.getResourceProperties("Index");
        HttpClient httpClient = new StdHttpClient.Builder().url(url).build();
        CouchDbInstance dbInstance = new StdCouchDbInstance(httpClient);
        CouchDbConnector db = dbInstance.createConnector(index, true);
        ObjectMapper mapper = new ObjectMapper();
        JsonNode node = mapper.readTree("{}");
        Map<String, Object> doc = mapper.convertValue(node, Map.class);
        doc = db.get(Map.class, username);
    }
```

Figure 4.8 Selection of documents from Elasticsearch and CouchDB databases

As shown in the Figure 4.8, Query API has been used in Elasticsearch taking various parameters in fields corresponding to searching, but in CouchDB we are making search through the user which is stored as an ID. This is due to the reason that we have Elasticsearch to auto-generate an Id particular to every document, which is unknown to the user deploying application. We are performing the selection process through the HTTP GET request.

For CouchDB: http://localhost:8080/CouchDB/readDocument/78(_id)

For Elasticsearch: http://localhost:8080/Elasticsearch/deleteDocument/78(user)

As shown in Figure 4.7, First GET operation is performed to retrieve the document to be updated and then POST operation is performed on the same document after required changes are made on it. Basically, Elasticsearch takes more time for reflection, whereas CouchDB performs it immediately.

For CouchDB: http://localhost:8080/CouchDB/updateDocument

For Elasticsearch: http://localhost:8080/Elasticsearch/updateDocument

```
//Updating a document in Elasticsearch database
    @RequestMapping(value = { "/updateDocument/{username}" }, method = {
            org.springframework.web.bind.annotation.RequestMethod.POST }, consumes = { "application/json" })
    @ResponseBody
    public DomainResponse updateDocument(@RequestBody String document) throws Exception {
        String url = ResourcePropertiesUtil.getResourceProperties("BaseUrl");
        ObjectMapper mapper = new ObjectMapper();
        JsonNode node = mapper.readTree(document);
        String user = node.get("user").asText();
        String newurl = url + "/_search?q=user:" + user;
        String searchresult = makeconnection(newurl, null, "GET");
        JsonNode jsonNode = mapper.readTree(searchresult);
        String id = jsonNode.path("hits").path("hits").get(0).path("_id").textValue();
        ObjectNode newDocument = (ObjectNode) mapper.readTree("{}");
        newDocument.put("college", node.get("college").textValue());
        newDocument.put("edu", node.get("edu").textValue());
        ObjectNode docNode = (ObjectNode) mapper.readTree("{}");
        docNode.put("doc", newDocument);
        url = url + "/" + id + "/_update";
        String result = makeconnection(url, docNode.toString(), "POST");
        }

//Updating a document in Elasticsearch database
{
        CouchDbConnector db = dbInstance.createConnector(index, true);
        ObjectMapper mapper = new ObjectMapper();
        JsonNode oldnode = mapper.readTree(document);
        Map<String, Object> olddoc = mapper.convertValue(oldnode, Map.class);
        JsonNode node = mapper.readTree("{}");
        Map<String, Object> doc = mapper.convertValue(node, Map.class);
        doc = db.get(Map.class, oldnode.get("user").asText());
        for (Entry<String, Object> jsonNode : olddoc.entrySet()) {
            if (doc.containsKey(jsonNode.getKey())) {
                doc.put(jsonNode.getKey(), jsonNode.getValue());
            }
        }
        db.update(doc);
}
```

Figure 4.9 Updation of document in Elasticsearch and CouchDB databases

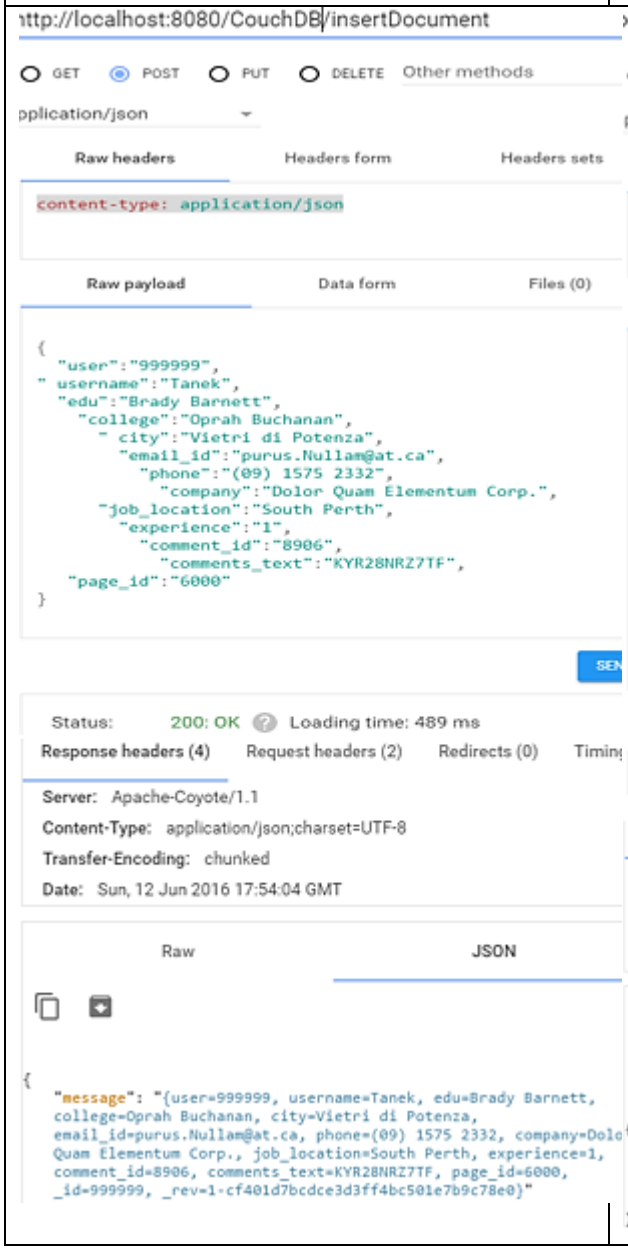
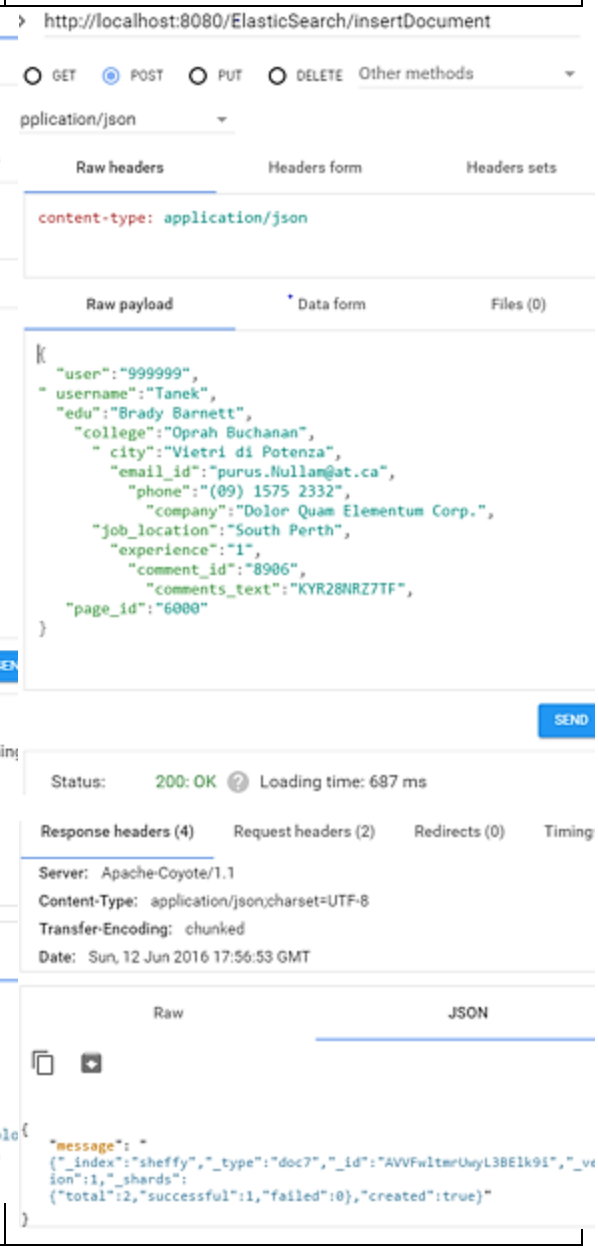
```
//Deleting the document from Elasticsearch Database
    @RequestMapping(value = { "/deleteDocument" }, method = {
            org.springframework.web.bind.annotation.RequestMethod.POST }, consumes = { "application/json" })
    @ResponseBody
    public DomainResponse deleteDocument(@RequestBody String document) throws Exception {
        String url = ResourcePropertiesUtil.getResourceProperties("BaseUrl");
        ObjectMapper mapper = new ObjectMapper();
        JsonNode node = mapper.readTree(document);
        String user = node.get("user").asText();
        String newurl = url + "/_search?q=user:" + user;
        String searchresult = makeconnection(newurl, null, "GET");
        System.out.println(searchresult);
        JsonNode jsonNode = mapper.readTree(searchresult);
        String id = jsonNode.path("hits").path("hits").get(0).path("_id").textValue();
        url = url + "/" + id;
        String result = makeconnection(url, document, "DELETE");
    }

//Deleting the document from CouchDB Database
{
    String url = ResourcePropertiesUtil.getResourceProperties("BaseUrl");
        String index = ResourcePropertiesUtil.getResourceProperties("Index");
        HttpClient httpClient = new StdHttpClient.Builder().url(url).build();
        CouchDbInstance dbInstance = new StdCouchDbInstance(httpClient);
        CouchDbConnector db = dbInstance.createConnector(index, true);
        ObjectMapper mapper = new ObjectMapper();
        JsonNode node = mapper.readTree("{}");
        Map<String, Object> doc = mapper.convertValue(node, Map.class);
        doc = db.get(Map.class, username);
        db.delete(doc);
    }
```

Figure 4.10 Deletion of document from Elasticsearch and CouchDB databases

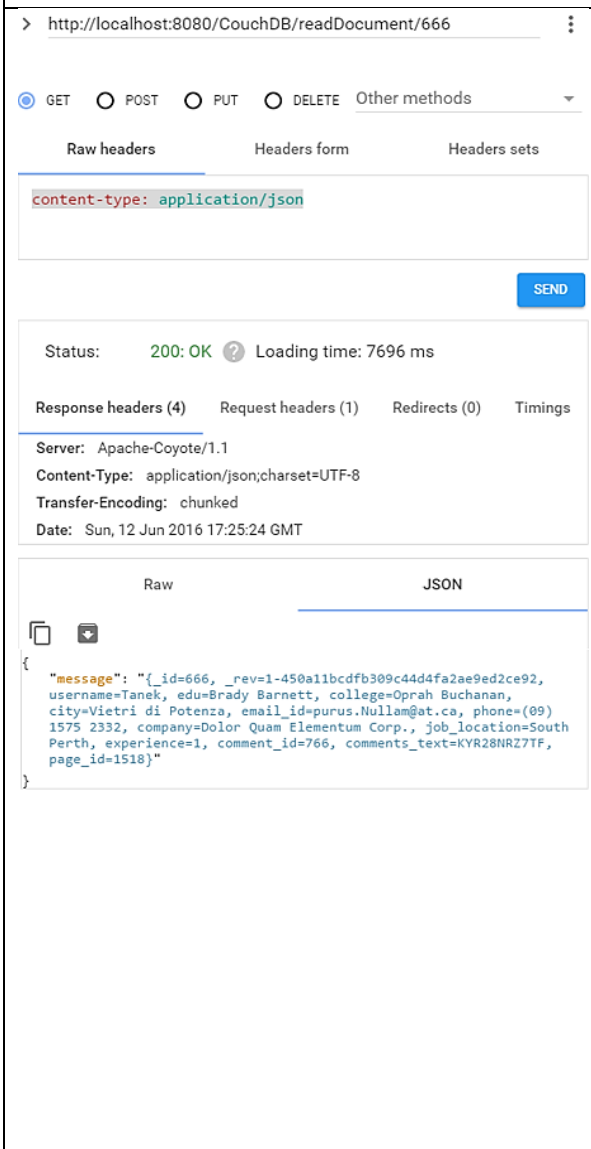
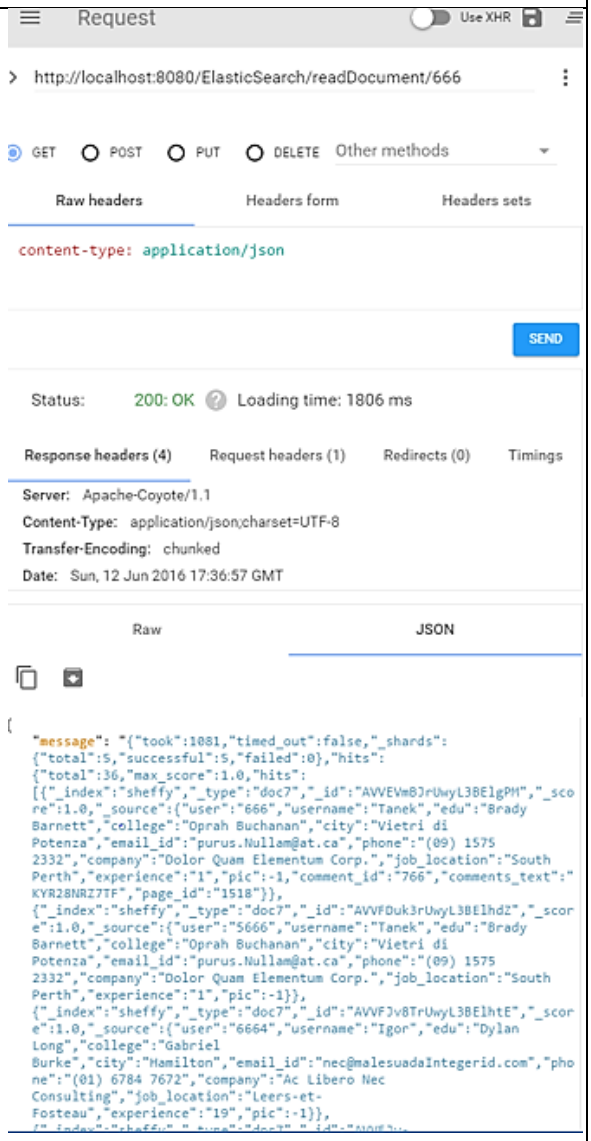
As shown Figure 4.10, for deleting a document from both databases, first document is fetched and corresponding Id is removed and it gets deleted from the database. The delete operation is performed through the HTTP GET request.

Table 4.2 CouchDB and Elasticsearch Insertion Query Results

CoucDB Insertion Query Results	Elasticsearch Insert Query Results
	

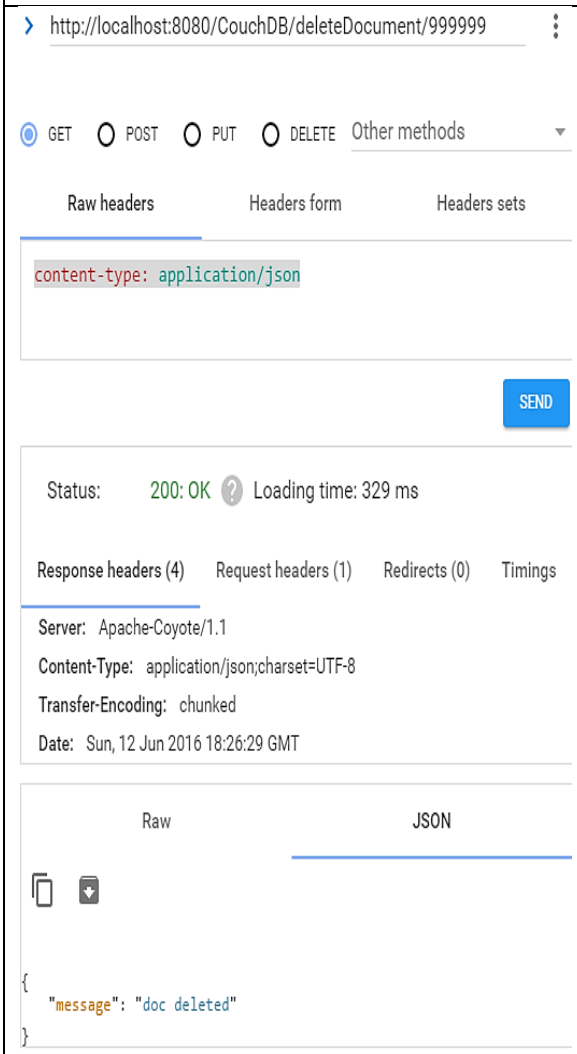
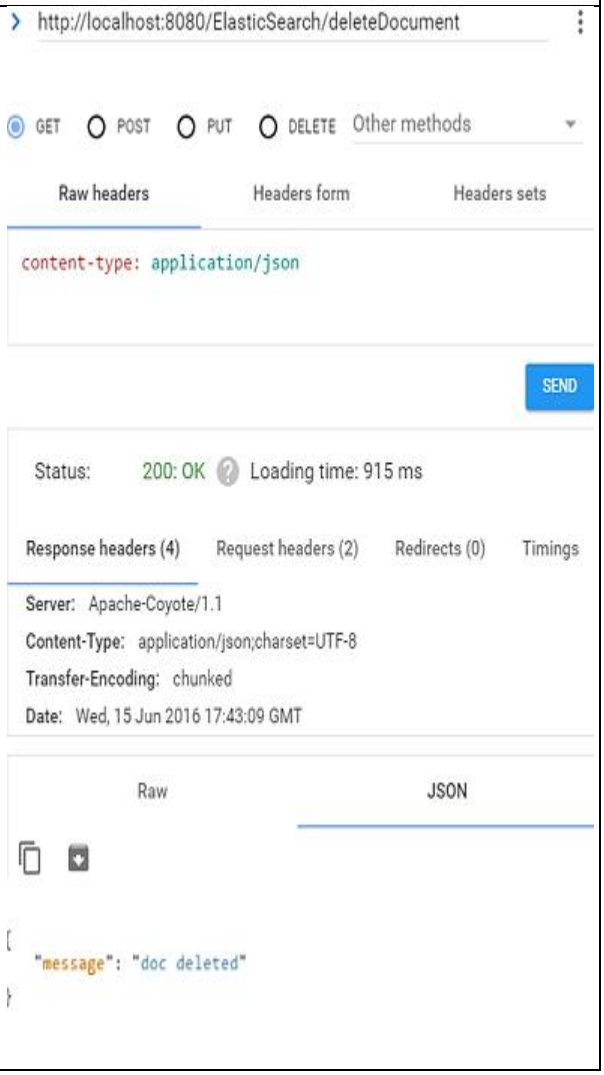
As shown in Table 4.2, execution of insertion operation in Elasticsearch takes much higher time than CouchDB. So, for storage purposes CouchDB is more efficient than Elasticsearch.

Table 4.3 CouchDB and Elasticsearch Selection Query Results

CoucDB Selection Query Results	Elasticsearch Selection Query Results
	

As shown in Table 4.3, the main difference in results of both the databases is during selection process. CouchDB retrieves only that particular document which is defined in the query, whereas in Elasticsearch, it retrieves all those documents that possess the substring of the document id provided in the query. It is shown clearly in the above table that Elasticsearch retrieves such large number of documents in much lesser time than CouchDB.

Table 4.4 CouchDB and Elasticsearch Delete Query Results

CoucDB Delete Query Results	Elasticsearch Delete Query Results
	

As shown in Table 4.4, Delete operation in CouchDB takes lesser time than Elasticsearch. So, the usage of databases highly depends upon the requirements of the user.

Chapter 5 Results and Comparative Analysis

In our research, we have compared the performance of two databases: Elasticsearch and Apache CouchDB. The tool used for implementation is Eclipse IDE. The code has been written using Java framework. The CouchDB, Elasticsearch and Maven java libraries have been used to make the connection and perform all the required operations for the comparison. As NoSQL handles unstructured data in a better way, we were motivated to perform the comparative analysis using datasets of image data. Rest Client API is used to hit the service of Elasticsearch and CouchDB. Insertion, Selection, Updation and Deletion operations have been performed and their respective results have been compared on the basis of time taken by both the databases to perform these operations. We have taken a sample dataset consisting of 20000 documents. The results of executed queries are shown in this section.

5.1 Analytical Comparison of CouchDB and Elasticsearch

The graphs and tables recorded by performing Bulk transfer, Insertion, Selection, Updation and Deletion operations on each of the databases (CouchDB and Elasticsearch) for performance evaluation are shown as follows.

Table 5.1 Bulk Data transfer Time from Relational to NoSQL Databases

Number of transfer operations	CouchDB(sec)	Elasticsearch(sec)
500	6.16	29.15
1000	11.74	56.22
5000	48.69	286.05
10000	93.23	516.17
15000	128.67	827.25
20000	147.14	1045.69

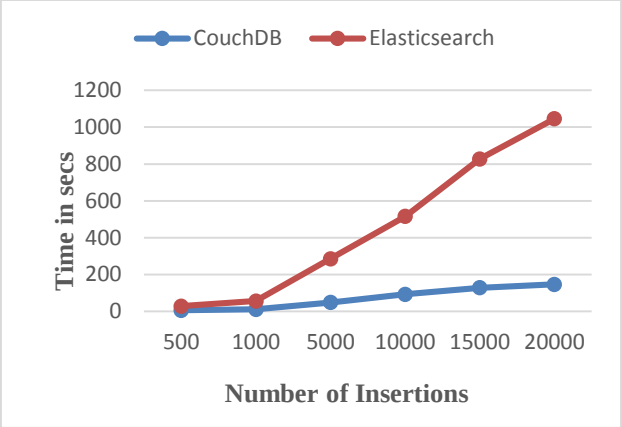


Figure 5.1 Bulk Data Transferring Time in Seconds

As shown in Figure 5.1, **x-axis** represents number of insertions and **y-axis** represents time taken in seconds. From this graph it is clear that the Elasticsearch has taken more time than CouchDB for inserting bulk data into the document inside the database during transferring data from Relational to NoSQL databases.

Table 5.2 Data Insertion Time (in milliseconds)

CouchDB(msec)	Elasticsearch(msec)
147	279
208	425
35	127
142	439
99	510
25	496
39	185
27	552
26	138
71	114

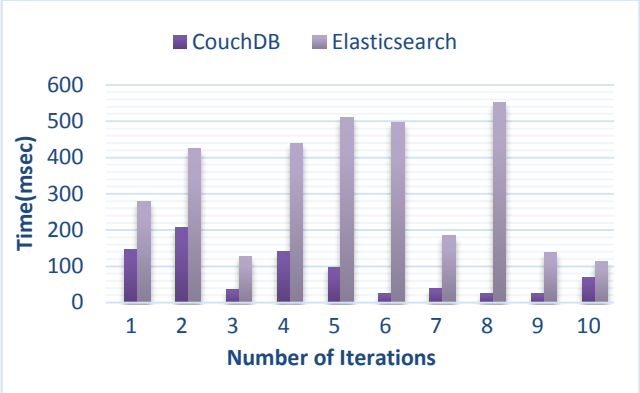


Figure 5.2 Graph representation of Insertion Time

As shown in Table 5.2 and Figure 5.2, x-axis represents number of iterations and y-axis represents the time taken in milliseconds. From this graph it is clear that the Elasticsearch has taken less time to retrieve the data from the database documents.

Table 5.3 Retrieval Time (in milliseconds)

CouchDB(msec)	Elasticsearch(msec)
240	275
170	115
151	95
89	24
154	51
225	84
183	46
85	36
116	39
110	60

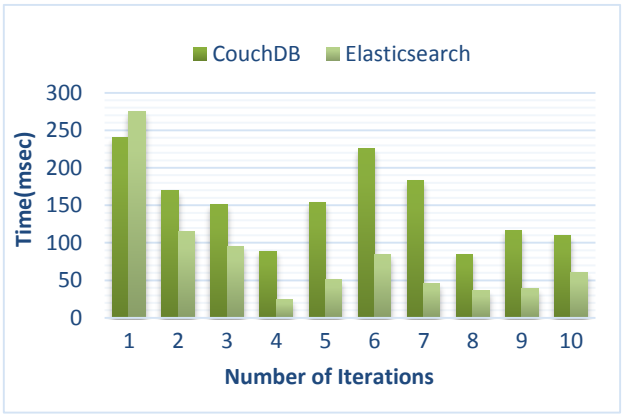


Figure 5.3 Graph representation of Retrieval Time

As shown in Figure 5.3, x-axis represents number of iterations and y-axis represents time taken in milliseconds. From this graph it is clear that the Elasticsearch has taken less time to retrieve the data from the database documents.

Table 5.4 Updation Time (in milliseconds)

CouchDB(msec)	Elasticsearch(msec)
330	430
46	328
47	125
124	181
29	175
28	231
24	144
77	164
155	297
62	248

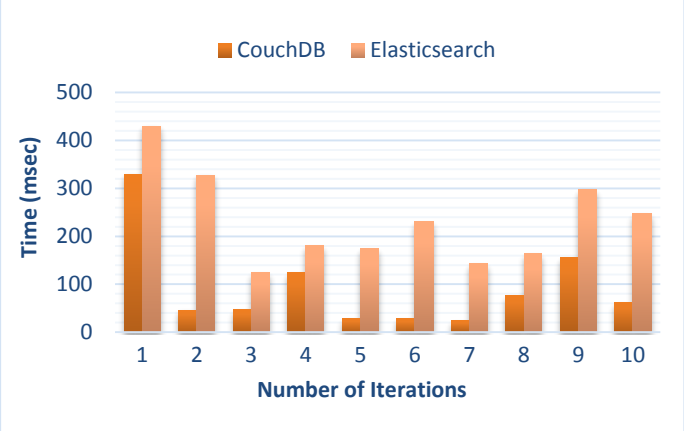


Figure 5.4 Graph representation of Updation Time

As shown in Figure 5.4, x-axis represents number of iterations and y-axis represents time taken in milliseconds. From this graph it is clear that the CouchDB has taken less time to update the data from the database documents.

Table 5.5 Deletion Time (in milliseconds)

CouchDB (msec)	Elasticsearch (msec)
401	616
110	408
167	533
69	389
114	472
68	166
84	376
101	584
115	551
240	352

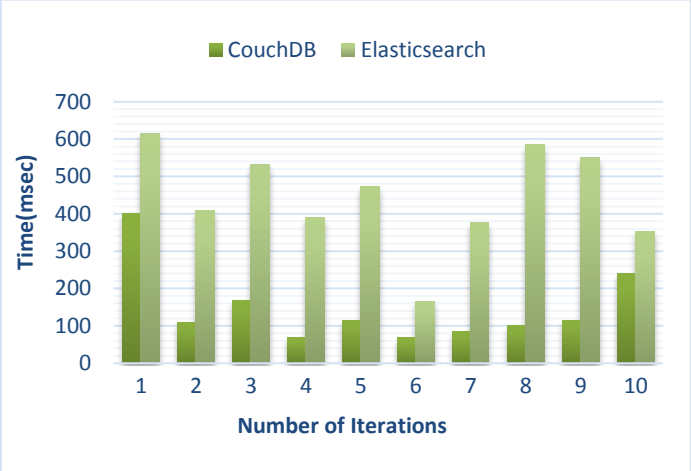


Figure 5.5 Graph representation of Deletion Time

As shown in Figure 5.5, x-axis represents number of iterations and y-axis represents time taken in milliseconds. From this graph it is clear that the CouchDB has taken less time to delete the data from the database documents.

Chapter 6 Conclusions and Future Work

There are various common features of distributed databases: Apache CouchDB and Elasticsearch and at the same time they possess certain differences also. CouchDB and Elasticsearch are NoSQL based document-oriented data models. In our thesis, we have discussed several features of these database systems and tried to understand why these databases are so widely used everywhere in the world and why companies are migrating their organizational data from the previous databases to these NoSQL databases.

Both these databases are used excessively for managing "Big Data". It solves basic problems faced by other database systems while handling big data sets. In our thesis we have implemented CouchDB and Elasticsearch on text datasets as well as on image datasets. The data taken is unstructured in nature and hence is more useful.

Table 6.1 Average time taken by Elasticsearch and CouchDB

	Insertion Time	Retrieval Time	Updation Time	Deletion Time
Elasticsearch	0.321 seconds	0.082 seconds	0.232 seconds	0.444 seconds
CouchDB	0.082 seconds	0.152 seconds	0.092 seconds	0.146 seconds

6.1 Conclusion

The following conclusions can be drawn from the results obtained.

i. CouchDB takes lesser time than Elasticsearch in regard of operations such as insertion, updation and deletion of dynamic data.

ii. In case of retrieval operation, CouchDB takes much higher time than Elasticsearch.

iii. The time taken by Elasticsearch to perform insertion, updation and deletion operations on unstructured data is more than double of that taken by CouchDB for the same and during selection operation, CouchDB takes more than double of that time taken by Elasticsearch.

iv. The selection of Data Models is highly dependent upon the preference of the user in terms of searching and other operations.

6.3 Future Work

We already have concluded that Elasticsearch is better in search and CouchDB is better in performing operations other than search. These databases are used for Business Intelligence leading to successful storage of data for analysis purposes. So as a future work, these databases can be integrated with ETL (Extract, Transform and Load) tools in order to analyze the live data possessing real-time behavior like data of real time applications and social networking sites. ETL is defined as a process in data warehousing that helps in extracting data from numerous homogeneous and heterogeneous resources and also transforms the data into required format for analysis and querying. Then through this, performance analysis of these databases on such “Big Data” projects can also be done.

References

[1] "CouchDB_in_the_wild - Couchdb Wiki", *Wiki.apache.org*, 2016. [Online]. Available: http://wiki.apache.org/couchdb/CouchDB_in_the_wild.

[2] D. Curry, C. Lawrence, T. Curwin, A. Razani and D. Power, "Why Large Hadron Collider Scientists are Using CouchDB - ReadWrite", *ReadWrite*, 2010. [Online]. Available: http://readwrite.com/2010/08/26/lhc-couchdb/.

[3] "These 15 Tech Companies Chose the ELK Stack Over Proprietary Logging Software", *Logz.io*, 2016. [Online]. Available: http://logz.io/blog/15-tech-companies-chose-elk-stack/.

[4] I.Hashem, I. Yaqoob, N. Anuar, S. Mokhtar, A.Gani and S. U. Khan, "The rise of "big data" on cloud computing: Review and open research issues", *Elsevier on Information Systems*, vol. 47, pp. 98-115, 2015.

[5] D. Singh and C. Reddy, " A survey on platforms for big data analytics", *Springer Journal of Big Data*, vol. 2, no. 1, pp. 1-20, 2015.

[6] J. Pokorny, "NoSQL databases: a step to database scalability in web environment", Emerald *International Journal of Web Information Systems*, vol. 9, no. 1, pp. 69-82, 2013.

[7] C. Gyorodi, R. Gyorodi, G. Pecherle and A. Olah, "A comparative study: MongoDB vs. MySQL", *in Proceedings of IEEE 13th International Conference on Engineering of Modern Electric Systems*, pp.1-6, 2015.

[8] A. Joshi, S. Haradhvala, and C. Lamb. "Oracle NoSQL databasescalable, transactional key-value store," in Proceedings of *2nd International Conference on Advances in Information Mining and Management*, pp 75-78, 2012.

[9] G. Wang and J. Tang, "The NoSQL Principles and Basic Application of Cassandra Model", *in Proceedings of IEEE International Conference on Computer Science and Service System*, pp. 1332-1335, 2012.

[10] V. Bhupathiraju and R. Ravuri, "The dawn of Big Data - Hbase", *in Proceedings of IEEE Conference on IT in Business, Industry and Government*, pp. 1-4, 2014.

[11] G. Wang and J. Tang, "The NoSQL Principles and Basic Application of Cassandra Model", *in Proceedinds of IEEE International Conference on Computer Science and Service System*, pp. 1332-1335, 2012.

[12] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden and I. Stoica, "BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data", *in Proceedings of the 8th ACM European Conference on Computer Systems*, pp. 29-42, 2013.

[13] S. Zhang, "Application of Document-Oriented NoSQL Database Technology in Web-based Software Project Documents Management System", *in Proceedings of IEEE Third International Conference on Information Science and Technology*, pp. 504-507, 2013.

[14] "'couchdb' tag wiki", *Codereview.stackexchange.com*, 2016. [Online]. Available: http://codereview.stackexchange.com/tags/couchdb/info.

[15] M. Ganiraju, M. A. Payton, J. A. Roth, L. V. Abruzzo, K. R. Coombes, "Relax with CouchDB — Into the non-relational DBMS era of bioinformatics", Elsevier journal of *Genomics*, vol. 100, no. 1, pp. 1-7, 2012.

[16] K. Kaur and R. Rani, "Modeling and Querying Data in NoSQL Databases", *in Proceedings of IEEE International Conference on Big Data*, pp. 1 – 7, Oct. 2013.

[17] N. Datt, "Comparative Study of CouchDB and MongoDB – NoSQL Document Oriented Databses", *International Journal of Computer Applications*, vol. 136, no. 3, pp.24-26, 2016.

[18] S. Burckhardt, "Principles of eventual consistency." *Foundations and Trends in Programming Languages*, vol. 1, no. 1-2,pp. 1-150, 2014.

[19] "Introduction_to_CouchDB_views - Couchdb Wiki", *Wiki.apache.org*, 2016. [Online]. Available: https://wiki.apache.org/couchdb/Introduction_to_CouchDB_views.

[20] M. Brown, *Getting started with CouchDB*. Sebastopol, CA: O'Reilly, 2012.

[21] R. Lerner, "At the forge: CouchDB views", *Linux Journal*, vol. 2010, no. 196, pp. 7, 2010.

[22] E. Meijer and G. Bierman, "A co-relational model of data for large shared data banks", *Communications of the ACM*, vol. 54, no. 4, pp. 49-58, 2011.

[23] J. Armstrong, "Erlang," Communications of the ACM, vol. 53, no.9, pp. 68-75, 2010.

[24] X. Li and Y. Wang, "Design and Implementation of an Indexing Method Based on Fields for Elasticsearch", in Proceedings of IEEE 5th International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC), pp. 626-630, 2015.

[25] O. Kononenko, O. Baysal, R. Holmes and M. Godfrey, "Mining modern repositories with Elasticsearch", in Proceedings of ACM 11th Working Conference on Mining Software Repositories, pp. 328-331, 2014.

[26] "Advantages of Elastic Search", 3Pillar Global, 2016. [Online].Available: http://www.3pillarglobal.com/insights/advantages-of-elastic-search.

[27] "Relativity Data Grid", *Help.kcura.com*, 2016. [Online]. Available: https://help.kcura.com/9.3/Content/Relativity/Data_Grid/Relativity_Data_Grid.htm.

[28] "Getting Started | Elasticsearch Reference [2.3] | Elastic", *Elastic.co*, 2016. [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html.

[29] R. Kuc, M. Rogozinski, A. Hussain, R. Youe, S. Srivastava and S. Siddiqui, "*Mastering elasticsearch", 2015*.

[30] J. Campbell, E. A. Santos and A. Hindle, "The unreasonable effectiveness of traditional information retrieval in crash report deduplication." In *Proceedings of ACM 13th International Workshop on Mining Software Repositories*, pp. 269-280, 2016.

[31] J. Zobel and A. Moffat, "Inverted files for text search engines", ACM Computing Surveys, vol. 38, no. 2, pp. 1–56, 2006.

[32] "Shield: You Know, For Security | Elastic", *Elastic Blog*, 2014. [Online]. Available: https://www.elastic.co/blog/shield-know-security-coming-soon.

[33] C. L P. Chen, and C. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on Big Data," Elsevier *Information Sciences, vol.* 275, pp. 314-347, 2014.

[34] Y. Huang, and T. Luo, "NoSQL Database: A Scalable, Availability, High Performance Storage for Big Data," Springer International *Pervasive Computing and the Networked World*, pp. 172-183, 2013.

[35] W. Fan and A. Bifet, "Mining Big Data: current status, and forecast to the future", ACM SIGKDD Explorations Newsletter, Vol. 14, no. 2, pp. 1-5, 2013.

[36] B. Purcell, "The emergence of "big data" technology and analytics," *Journal of Technology Research, vol. 4, pp. 1-7,* 2013.

[37] E. Bertino, D. Jewell, M. Hammersley and Y. Li, "Performance and Capacity Implications for Big Data", IBM Redbooks, 2014

[38] S.V. Phaneendra and E.M. Reddy, "Big Data-solutions for RDBMS problems-A survey," *IEEE International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 9, September 2013.

[39] J. Lee, S. H. Sang, and M. J. Lee, "Issues in developing object-oriented database systems for real-time applications," In Proceedings of IEEE workshop on *Real-Time Applications, pp. 136-140, 1994*.

[40] C. Tyagi and S. Batra, "Comparative analysis of relational and graph databases," *International Journal of Software Computing and Engineering (IJSCE), vol. 2, no. 2, pp.* 509-512, 2012.

[41] K. Grolinger, W. A. Higashino, A. Tiwari and M. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," Springer *Journal of Cloud Computing: advances, systems and applications* Vol. 2, no. 1, 2013.

[42] R. Arora and R. Rani, "An Algorithm for Transformation of Data from MySQL to NoSQL (MongoDB)," *International Journal of Advanced Studies in Computers, Science and Engineering (IJASCSE),* vol 2, no. 1, pp. 6-12,2013.

[43] "Butler Group Report: Enterprise Search and Retrieval - Unlocking the Organisation's Potential", *Ithound.com,* 2016. [Online]. Available: http://www.ithound.com/abstract/butler-report-enterprise-search-retrieval-unlocking-organisation-potential-290.

[44] "Enterprise Search Surveys", *Searchtechnologies.com,* 2016. [Online]. Available: http://www.searchtechnologies.com/enterprise-search-surveys.

[45] Q. Liping and W. Lidong. "An evaluation of Lucene for keywords search in large-scale short text storage," in Proceedings of IEEE International Conference on *Computer Design and Applications (ICCDA),* vol. 2, pp. 206-209, 2010.

[46] Y. Zhang and J. Li, "Research and improvement of search engine based on Lucene," In Proceedings of IEEE *International Conference on Intelligent Human-Machine Systems and Cybernetics*, pp. 270-273, 2009.

[47] Mathew, A. Brigit, P. Pattnaik, and S. D. M. Kumar, "Efficient information retrieval using Lucene, LIndex and HIndex in Hadoop," In Proceedings of IEEE 11th International Conference on *Computer Systems and Applications (AICCSA), pp. 333-340,* 2014.

[48] K. Lee, Y. Lee, H. Choi, Y. D. Chung and B. Moon, “Parallel Data Processing with MapReduce: A Survey,” *ACM SIGMOD Record*, vol. 40, no. 4, pp. 11-20, December 2011.

[49] H. M. L. Dharmasiri and M.D.J.S. Goonetillake "A Federated Approach on Heterogeneous NoSQL Data Stores", in Proceedings of IEEE *International Conference on Advances in ICT for Emerging Regions*, pp. 234 - 239, 2013.

[50] R. Cattell, “ Scalable SQL and NoSQL data stores,” ACM SIGMOD Record, vol. 39, no. 4, pp. 12-27, 2010.

[51] Z. Liu, Y. Wang, R. Lin,” A novel development and analysis solution to PaaS log by using CouchDB." In proceedings of *3rd IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC), pp*. 251-255, 2012

[52] L. Ramaswamy, V. Lawson, S. V. Gogineni,” Towards a quality-centric big data architecture for federated sensor services,” *in IEEE International Conference on Big Data*, pp. 86-93, 2013

[53] D. Havlik, M. Egly, H. Huber, P. Kutschera, M. Falgenhauer, M. Cizek, “Robust and trusted crowd-sourcing and crowd-tasking in the Future Internet,” in *Environmental Software Systems. Fostering Information Sharing*, vol. 413, pp. 164-176,2013.

List of Publications and Video Link

Publications

S. Gupta and R. Rani, **"***Data Transformation and Query Analysis of Elasticsearch and CouchDB Document Oriented Databases***"**, IEEE Conference on Inventive Computation Technologies (ICICT 2016). Date-26-27 August-16 [ ]

Video Link

https://youtu.be/-WYzMkpwklA

Plagrism Report

spaper

ORIGINALITY REPORT

7%	6%	2%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	dspace.thapar.edu:8080 Internet Source	2%
2	www.mongodb.com Internet Source	2%
3	Dharmasiri, H. M. L., and M. D. J. S. Goonetillake. "A federated approach on heterogeneous NoSQL data stores", 2013 International Conference on Advances in ICT for Emerging Regions (ICTer), 2013. Publication	<1%
4	www.journalofcloudcomputing.com Internet Source	<1%
5	ethesis.nitrkl.ac.in Internet Source	<1%
6	Smart Innovation Systems and Technologies, 2016. Publication	<1%
7	webdam.inria.fr Internet Source	<1%