©2010 International Journal of Computer Applications (0975 – 8887)
Volume 1 – No. 13

Moving Towards Non-Relational Databases

Uma Bhat
Usha Mittal Institute of Technology,
SNDT Women's University,
Santacruz (W), Mumbai, 400049

Shraddha Jadhav
Usha Mittal Institute of Technology,
SNDT Women's University,
Santacruz (W), Mumbai, 400049

This paper is a part of the semester-long project work undertaken in the final year of B.Tech. Course. It was carried out under the aegis of Patni Computer Systems Ltd., under the guidance of Sunil Joglekar, Senior Technical Architect, Product & Technology Initiative Group.

ABSTRACT
RDBMS has been around since long. It is the founder stone of many application stacks. It provides the users with the best mix of simplicity, robustness, flexibility, performance, scalability, and compatibility. However the emergence of the cloud centric application poses a set of challenges to the existing RDBMS Vendors. The RDBMS are not so suitable to cater to some of the critical requirement of these new generation applications such as handling large set of unstructured data or providing elastic scalability. This results in emergence of a new set of document centric or resource centric databases which are non-relational in nature. Promising non-relational solutions include CouchDB, SimpleDB etc. We have explored and conducted few experiments with some of such databases.

In the current paper we would like to highlight the salient features of these databases. With the help of examples we would describe how these databases differ from the conventional relational databases. We would also discuss how they cater to the requirement of today's modern enterprise applications

Categories and Subject Descriptors
H.2.0 [**Database Management**]: Security, Integrity and Protection

H.2.1 [**Logical Design**]: Data Models, Normal forms, Schema and Sub schema

E.5 [**Files**]: Backup Recovery, Optimization, Searching/Sorting, Organization/Structure

General Terms

Performance, Reliability, Security

Keywords
Non-Relational Databases, CouchDB, SimpleDB, Bigtable, Data Integrity, RESTful JSON, Document-Oriented, Schema-free, Map/Reduce, Replication, Tolerance, Consistency, Scalability.

1. INTRODUCTION
Database model is a theory or specification describing how a database is structured and used. Several such models have been suggested such as hierarchical, network, relational and non-relational. Now-a-days, relational database models are the dominant persistent storage technology. Inspite of this fact, it has many shortcomings which can hinder performance levels. As more and more applications are launched in environments that have massive workloads such as cloud and web services, their scalability requirements change very quickly and also grow very large. It is difficult to manage with a relational database sitting on a single in-house server. To overcome all these shortcomings, vendors can opt for non relational database models.

Non-Relational databases enjoy schema-free architecture and possess the power to manage highly unstructured data. They can be easily deployed to multi-core or multi-server clusters serving modularization, scalability and incremental replication. Non relational databases being extremely scalable, offer high availability and reliability, even while running on hardware that is typically prone to failure, thereby challenging relational database, where consistency, data integrity, uptime and performance are of prime importance.

In this paper with the help of a case study, we have attempted to demonstrate how non - relational databases such as, Apache CouchDB, "Cluster Of Unreliable Commodity Hardware", Google's Big Table and Amazon SimpleDB would prove to be tried, tested and trusted solutions overcoming drawbacks of relational databases.

2. CHALLENGES WITH RELATIONAL DATABASES

a) Performance issues are difficult to predict

While working with a shared database the performance characteristics of database is hard to predict because each application accesses the database in its own unique way.

b) Data integrity is difficult to ensure with shared databases

Because no single application has control over the data it is very difficult to be sure that all applications are operating under the same business principles.

c) Operational databases require different design strategies than reporting databases

The schemas of operational databases reflect the operational needs of the applications that access them, often resulting in a reasonably normalized schema with some portions of it denormalized for performance reasons. Reporting databases are highly denormalized with significant data redundancy within them to support a wide range of reporting needs.

3. PROMISING NON RELATIONAL DATABASES

Web 2.0 approaches and design patterns are becoming more established in online applications and enterprise architecture. Social architectures, crowdsourcing, and open supply chains are becoming the norm in the latest software systems faster than expected in many cases. New advances in processors, virtualization technology, disk storage, broadband Internet access and fast, inexpensive servers have all combined to make cloud computing a compelling paradigm.

Unfortunately, as a result, the architectural expertise needed to effectively leverage these ideas is often far from abundant. Non relational databases like Apache CouchDB, Amazon's SimpleDB and Google's Bigtable prove to be effective methods in overcoming these shortcomings.

3.1 Apache CouchDB: Overview

CouchDB is an open source document-oriented database-management system, accessible using a RESTful JavaScript Object Notation (JSON) API. The term "Couch" is an acronym for "Cluster Of Unreliable Commodity Hardware," reflecting the goal of CouchDB being extremely scalable, offering high availability and reliability, even while running on hardware that is typically prone to failure. CouchDB was originally written in C++ but moved

to the Erlang OTP platform for its emphasis on fault tolerance. It is a database built for the future. CouchDB has been developed from the ground up with Web applications as the primary focus and has its sights on becoming the de-facto database for Web application development.

3.1.1 Features CouchDB

a) Schema Free

CouchDB has a hand full of advantages over relational database model and also aims at storing data in documents, which have no schema. One document can have a field that another one does not have. The documents in CouchDB are actual representations of the data objects. CouchDB stores data in the JSON format. It is language-agnostic. Data is serialized and de-serialized to and from that format. JSON can include all the native data types in a programming language.

b) Document Oriented Structure

Document oriented structure forms building blocks of CouchDB database. CouchDB databases store uniquely named documents with document ID and revision number. All data in a CouchDB database is stored in a document, and each document can be made up of an undefined number of fields which are not bound by size and have unique names. Documents can have attachments which can be of both text as well as digital format. When changes are made to CouchDB document a new version of the entire document called a revision is created. The document-revision system works in much the same way as a wiki or Web-based document management system manages revision control. CouchDB does not feature locking mechanisms; two clients can load and edit the same document at the same time. CouchDB maintains data consistency by ensuring that document updates are all or nothing — it either works or it fails.

c) Concurrent

CouchDB is written in erlang. Erlang is a functional programming language, a virtual machine, and a set of standard libraries. CouchDB uses Erlang because the problems that it solved for Telecom applications are the same for the Web today. CouchDB has an inbuilt ACID compliant datastore which is referred to as Non–locking Multi Version Concurrency Control. Concurrent requests can continue to be served by using versioning and it never overwrites data. Compaction can be used to get rid of previous revisions and to reclaim disk space thus making it crash-resistant

d) RESTful HTTP API

CouchDB treats all stored items in the database as resources. It offers an API as a means to retrieve data from the database. This API is accessible via HTTP `GET` and `POST` requests, and returns data in the form of JavaScript objects using JSON. HTTP is understood, interoperable, scalable and proven technology and can be used with software and hardware for caching, proxying and load balancing.

Four basic operations on document using CouchDB via HTTP REST API:

Create: HTTP PUT /db/docid
Read: HTTP GET /db/docid
Update: HTTP POST /db/docid
Delete: HTTP DELETE /db/docid

It makes use of simple HTTP create, request, update and delete operations on documents. It is accessible just by giving HTTP requests through the browser.

e) Map/Reduce

The Map and Reduce functions of Map/Reduce are both defined with respect to data structured in (key, value) pairs. Map() is a user defined function which transform each documents into zero, one or multiple intermediate objects, reduce() is user defined function to consolidate the intermediate objects into the final result.

f) Replication

CouchDB achieves eventual consistency between databases by using incremental replication, a process where document changes are periodically copied between servers. Share nothing clusters of databases can be built where each node is independent and self-sufficient, leaving no single point of contention across the system. Replication allows synchronizing two or more CouchDB instances and includes automatic conflict detection and resolution. When trying to replicate, CouchDB detects the conflict and each node independently chooses one conflicting revision to be the winning revision. All losing revisions get stored as previous revisions.

3.1.2 Data Model for COUCHDB

{
"_id":"44e55a9a3ebe5204a747343b9134",

"_rev": "11-4290911434",
"clinical history": {"Kidney stone": "yes", "TB": "no"},
"name": "Mr. Kiran",
"symptoms": {"1": "Vomiting", "2": "Fever", "3": "Stomach_ache"},
"personal_data": {"Age": 36, "Height": 179, "Weight": 76},
"test": {"a": "Culture_test", "b": "Blood_test", "c": "HC/CBC"}
}

3.1.3 Advantages

a) Scalability
CouchDB embraces RESTful architectural constraints to promote resolvability, efficiency, performance, and reliability thus enhancing scalability.

b) Schema Free
There is no need to predefine a schema before creating documents. All documents can be independent which reduces their inter dependency and there is no need to update database when a field is added.

c) HTTP request-response mechanism
It makes use of simple HTTP create, request, update and delete operations on documents. It is accessible just by giving HTTP requests through the browser.

d) Support for attachments
It stores file attachments in the form of images, music, and flash. Thus, it supports digital attachments which are not seen in traditional relational databases.

e) Zero configuration Replication
Now, there is no need to carry storage devices. Data can be replicated in the network across nodes both in local and remote fashion with absolutely no need of internet connection.

f) Provision of Web administration interface-Futon
Futon JavaScript interface can be used for displaying and editing data, deleting, inserting, creating, updating documents and triggering replication.

g) Low Memory requirement
Takes 150MB compared to 8Gig taken by MySQL for a similar database set-up.

3.1.4 Difference between RDBMS and COUCHDB

a) Unstructured with no Table-Format
Relational databases are structured and have a predefined schema. But CouchDB is a document oriented database where data is stored in the document itself, not in a related table as it would be in a relational database. There are no tables, rows, columns or relationships in a document-oriented database at all.

b) Schema Free
No strict schema needs to be defined in advance before using the database. If a document needs to add a new field, it can simply include that field, without adversely affecting other documents in the database. These documents do not have to store empty data values for fields they do not have a value for.

c) Concept of Identifiers
Relational databases use concept of primary keys, generated by an auto-increment feature or by a sequence generator. These identifiers are unique for the table or database they are used on, hence they can be reused by other tables and databases. If an update operation is made at the same time on two databases on separate networks, they cannot both accurately retrieve the next unique identifier. CouchDB does not come with an auto-increment or sequence feature. Instead, it assigns a Universally Unique Identifier (UUID) to each and every document, making it almost impossible for another database to accidentally select the same unique identifier.

d) Concept of views
Document-oriented databases do not support joins. This is a consequence of there being no primary and foreign keys in CouchDB. Instead it provides a feature called view which allows creating an arbitrary relation between documents that is not actually defined in the database itself. This means all benefits of typical SQL join queries can be achieved without the burden of predefining their relationships in the database layer.

e) Concept of Identifiers
Relational databases use concept of primary keys, generated by an auto-increment feature or by a sequence generator. These identifiers are unique for the table or database they are used on, hence they can be reused by other tables and databases. If an update operation is made at the same time on two databases on separate networks, they cannot both accurately retrieve the next unique identifier. CouchDB does not come with an auto-increment or sequence feature. Instead, it assigns a Universally Unique Identifier (UUID) to each and every document, making it almost impossible for another database to accidentally select the same unique identifier.

f) Replication
CouchDB uses replication to propagate application changes across participating nodes. This is a fundamentally different approach from relational databases, which operate at different intersections of consistency, availability, and partition tolerance. This can be illustrated by the following diagram.

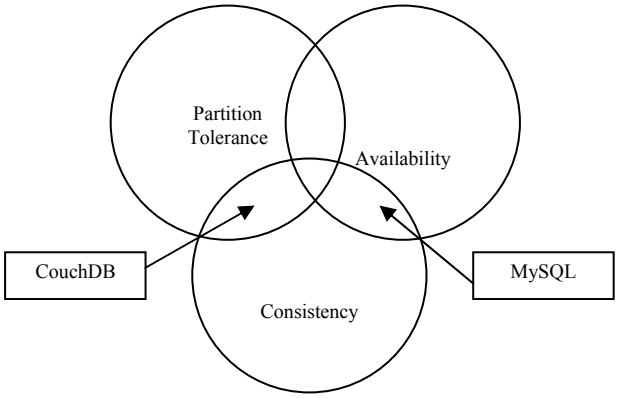

Fig 1. Difference between CouchDB and RDBMS

- **Consistency**

All database clients see the same data, even with concurrent updates.

- **Availability**

All database clients are able to access some version of the data.

- **Partition tolerance**

The database can be split over multiple servers.

3.2 Amazon SIMPLEDB: Overview

Amazon SimpleDB is a web service providing the core database functions of data indexing and querying. This service works in close conjunction with Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2), collectively providing the ability to store, process and query data sets in the cloud, making web-scale computing easier and more cost effective for developers. A traditional, clustered relational database requires a sizable upfront capital outlay, is complex to design, and often requires a DBA to maintain and administer. Amazon SimpleDB is simpler, requiring no schema, automatically indexing and providing a simple API for storage and access. This approach eliminates the administrative burden of data modeling, index maintenance, and performance tuning. Developers gain access to this functionality within Amazon's proven computing environment, are able to scale instantly, and pay only for what they use.

3.2.1 Features of SimpleDB

a) Simple to use

Amazon SimpleDB provides streamlined access to the lookup and query functions that traditionally are achieved using a relational database cluster while leaving out other complex, often-unused database operations. The service allows you to quickly add data and easily retrieve or edit that data through a simple set of API calls.

b) Flexible

With Amazon SimpleDB, it is not necessary to pre-define all of the data formats to be stored; simply new attributes can be added to the Amazon SimpleDB data set when needed, and the system will automatically index the data accordingly. The ability to store structured data without first defining a schema provides developers with greater flexibility when building applications.

c) Scalable

Amazon SimpleDB allows to easily scale the application. New domains can be quickly created as data grows or request throughput increases. Currently, store up to 10 GB per domain can be stored and up to 100 domains can be created.

d) Fast

Amazon SimpleDB provides quick, efficient storage and retrieval of data to support high performance web applications.

e) Reliable

The service runs within Amazon's high-availability data centers to provide strong and consistent performance. To prevent data from being lost or becoming unavailable, fully indexed data is stored redundantly across multiple servers and data centers.

f) Low touch

Accessing capabilities through the AWS cloud eliminates the complexity of maintaining and scaling operations in-house. The service allows focusing on value-added application development, rather than arduous and time-consuming database administration.

g) Designed for use with other Amazon Web Services

Amazon SimpleDB is designed to integrate easily with other web-scale services such as Amazon EC2 and Amazon S3.

h) Inexpensive

Amazon SimpleDB passes on the financial benefits of Amazon's scale. Vendors have to pay only for resources they consume.

Data Model for SimpleDB

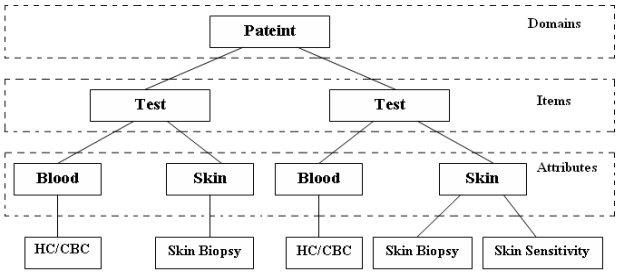

Fig 2. A SimpleDB model

3.2.2Advantages

a) No big infrastructure investment

Freedom from the shackles of big infrastructure investment opens up great opportunities for innovation. Vendors can now focus on their business ideas instead of fretting over the number of servers they have, worrying about running out of disk space, etc. Amazon worries about the mundane details of the hardware and infrastructure and how to make it highly available while they can concentrate on bringing their ideas to life. Low setup costs and pay-as-you-go expansion make it perfect for startups.

b) Maintenance is simpler

Setting up and maintaining a highly available clustered database that is constantly growing is extremely difficult. However,

maintenance in SimpleDB is much simpler than a typical database because there is nothing to set up or configure. Amazon takes care of all the administrative tasks. Data is automatically indexed by Amazon and is available anytime from anywhere.

c) High Scalability

SimpleDB is designed from the ground up to address the use case of scaling massive amounts of data utilizing a cloud approach. All data stored in SimpleDB is replicated multiple times in geographically disbursed data centers, so customer databases need not be backed and will automatically fail over to another replica if one is not available. Requests can be done via https for encryption.

d) Parallelism

It is based on Erlang which supports both distributed and concurrent operations. Hence, data are stored across multiple nodes which support parallel query execution

e) Schema free

The ability to store structured data without first defining a schema provides developers with greater flexibility when building applications and eliminates the need to re-factor an entire database as those applications evolve.

f) Better storage

SimpleDB can be used as both data storage and an indexing service. For example, one can use SimpleDB as a flat-file store, where each record maps onto a line. Another use case is to use each record in SimpleDB as Meta data for a media file that is stored on Amazon S3. This solution enables quick search of media files via SimpleDB and then retrieval via Amazon S3. Individual item names, attribute names, and attribute values can be up to 1,024 bytes in length. Amazon SimpleDB allows 10GB of storage for each domain with 100 domains per customer account, which provides you with 1 TB of total storage.

3.2.3 Differences between RDBMS and SIMPLEDB

a) Items are stored in hierarchical structure, not a table

Items stored in SimpleDB domain can contain multiple attributes, each of which may have multiple values. Relationship between these resources is best visualized as a hierarchical tree structure rather than as a rigid, predefined table structure. Because, there are no predefined database or table schemas, an item can have a different set of attributes from the other items in a domain. There is freedom to rearrange attribute and value portions of the tree as and when new data elements are added.

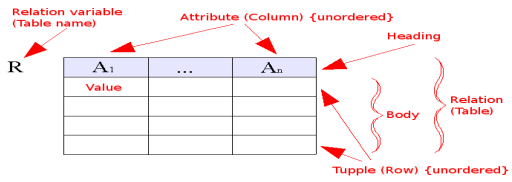

Fig 3. RDBMS Model

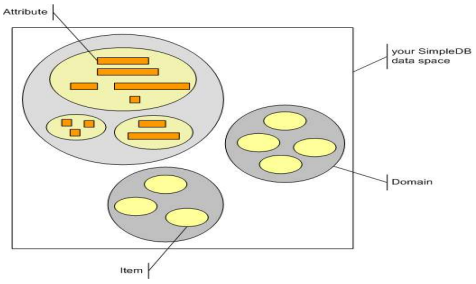

Fig 4. SimpleDB Model

b) Better querying capabilities

Amazon SimpleDB does not storing raw data. It takes data as input and expands it to create indices across multiple dimensions to quickly query that data. Amazon SimpleDB stores smaller bits of data and uses dense drives that are optimized for data access speed.

c) Provision of technical benefits not offered by RDBMS

Amazon SimpleDB requires no schema, automatically indexes your data and provides a simple API for storage and access. This eliminates the administrative burden of data modeling, index maintenance, and performance tuning. Developers gain access to this functionality within Amazon's proven computing environment, are able to scale instantly, and pay only for what they use. Traditionally, this type of functionality has been accomplished with a clustered relational database that requires a sizable upfront investment, brings more complexity than is typically needed, and often requires a DBA to maintain and administer. In contrast, Amazon SimpleDB is easy to use and provides the core functionality of a database, real-time lookup and simple querying of structured data without the operational complexity.

3.3 Google's Big Table: Overview

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: terabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products.

Google's Appengine is built on top of DataStore, which is built on top of Bigtable.

Google App Engine datastore

The App Engine datastore is a schemaless object datastore, with a query engine and atomic transactions. The Python interface includes a rich data modeling API and a SQL-like query language called GQL.

The Google App Engine datastore provides robust scalable data storage for your web application. The datastore is designed with web applications in mind, with an emphasis on read and query performance. It stores data entities with properties, organized by application-defined kinds. It can perform queries over entities of

the same kind, with filters and sort orders on property values and keys. All queries are pre-indexed for fast results over very large data sets. The datastore supports transactional updates, using entity groupings defined by the application as the unit of transactionality in the distributed data network.

3.3.1 Features

a) Schema Free
The App Engine datastore is a schemaless object datastore, with a query engine and atomic transactions. The Python interface includes a rich data modeling API and a SQL-like query language called GQL.

b) Scalable
The datastore uses a distributed architecture to manage scaling to very large data sets.

c) Queries and Indexes
An App Engine datastore query operates on every entity of a given kind (a data class). App Engine has defined Query and GqlQuery class to query the datastore. The Query class is a datastore query interface that uses objects and methods to prepare queries. The GqlQuery class is a datastore query interface that uses the App Engine query language GQL. GQL is a SQL-like query language suitable for querying the App Engine datastore. An entity is returned as a result for a query if the entity has at least one value (possibly null) for every property mentioned in the query's filters and sort orders, and all of the filter criteria are met by the property values.

d) Consistency and Reliability Of Data
With the App Engine datastore, every attempt to create, update or delete an entity happens in a transaction. A transaction ensures that every change made to the entity is saved to the datastore, or, in the case of failure, none of the changes are made. This ensures consistency of data within an entity. The datastore can execute multiple operations in a single transaction, and roll back the entire transaction if any of the operations fail. This is especially useful for distributed web applications, where multiple users may be accessing or manipulating the same data object at the same time.

e) Quotas and Limits
Each call to the datastore API counts toward the Datastore API Calls quota. Note that some library calls result in multiple calls to the underlying datastore API.
Data sent to the datastore by the app counts toward the Data Sent to (Datastore) API quota. Data received by the app from the datastore counts toward the Data Received from (Datastore) API quota. The total amount of data currently stored in the datastore for the app cannot exceed the Stored Data (adjustable) quota. This includes entity properties and keys, but does *not* include indexes.

3.3.2 Data Model for Google's Datastore

```
class Patient(db.Model):

id = db.IntegerProperty()
_rev= db.StringProperty()
clinicalHistory= db.StringProperty()
name = db.StringProperty()
symptoms= db.ReferenceProperty(Symptom)
persoanlData= db.ReferenceProperty(PersonalData)
test= db.ReferenceProperty(Test)

where Symptom ,PersonalData and Test are other entities.

class Symptom(db.Model):

id = db. IntegerProperty()
symptom = db.StringProperty()
class PersonalData(db.Model):

age= db.Integer Property()
symptom = db.StringProperty()
class Test(db.Model):

id= db.Integer Property()
test_name = db.StringProperty()
```

3.3.3 Advantages

a) Better Scaling
Unlike traditional databases, the datastore uses a distributed architecture to manage scaling to very large data sets. An App Engine application can optimize how data is distributed by describing relationships between data objects, and by defining indexes for queries.

b) Better Design
The App Engine datastore is strongly consistent, but it's not a relational database. While the datastore interface has many of the same features of traditional databases, the datastore's unique characteristics imply a different way of designing and managing data to take advantage of the ability to scale automatically.

c) Schema Free
The App Engine datastore is a schemaless object datastore, with a query engine and atomic transactions. This provides the flexibility in managing structured data that is designed to scale to a very large size: terabytes of data across thousands of commodity servers.

3.3.4 Differences between RDBMS and Google's Bigtable

a) Scaling
Unlike traditional databases, the datastore uses a distributed architecture to manage scaling to very large data sets. An App Engine application can optimize how data is distributed by describing relationships between data objects, and by defining indexes for queries.

b) Uniqueness Of Data
The concept of Primary Key in Relational Database ensures uniqueness of data. There is no such concept in Google's DataStore. Datastore does not provide any implicit method to maintain uniqueness. The only thing that is unique is key/key_name across all your entities. However, there have been a couple of methods (individual projects) that provide this feature. If you don't use above framework, or create something of your own to maintain uniqueness, AppEngine has no problem with

Sduplicate entries, since the entries are not duplicate. Every entity has a unique key/key name.

c) Relationship between entities

The concept of foreign key, helps us in establishing one-one, one-many, many-many relationship between various entities. However, there is no such concept in Datastore. For implementing something similar to foreign key, you need what is called Reference Property. Again, reference property is nothing like foreign key. So, don't expect an error when an entity is deleted and the other entity still points to the non-existing entity.

4. COMPARISON BETWEEN NON-RELATIONAL DATABASES

4.1 Scorecard for non-relational databases

Table 1. Performance Matrix

	Capability	Manageability	Availability	Scalability	Value	Overall Space
%	25	25	20	20	10	
Apache CouchB	7	7	8	7	9	7.4
%	25	25	20	20	10	
Amazon SimpleDB	8	8	8	9	8	8.2
%	25	25	20	20	10	
Google Bigtable	8	8	8	9	8	8.2

5. CONCLUSION

Thus, through various illustrations given in this paper, we have tried to showcase features of non relational databases which ensure that better integrity, scalability, robustness are achieved. Enhanced data modeling and representation, faster computations, load balancing are the major targets of using these database concepts. They can bring about a revolution in this modern era of supercomputer power through their resource centric and support for versatile data approaches.

6. ACKNOWLEDGMENTS

Creation of a report requires great effort, hard work and blessings of many great people who directly or indirectly contribute to the report. This report is no exception and we owe special gratitude to several persons.

First and foremost it is a matter of great pleasure to express our gratitude to Usha Mittal Institute of Technology for granting us this lifetime experience which we shall always treasure. Also, we are extremely indebted to Patni Computer Systems for giving us an opportunity to explore a new dimension of the upcoming technologies and providing us this brilliant opportunity.

Last but not the least we would like to thank our parents, friends, lab assistants for supporting in our endeavors.

7. REFERENCES

1) Joe Lennon," Exploring CouchDB-A document-oriented database for Web applications", 31 Mar 2009 [Online]. Available: http://www.ibm.com/developerworks/opensource/library/os-couchdb/ [Accessed: March 12, 2009].
2) "Homepage – CouchDB", Apache Software Foundation, 2008 [Online]. Available: http://couchdb.apache.org/ [Accessed: March 3, 2009].
3) Koswik, "Interactive CouchDB", 3 April, 2009 [Online]. Available: http://labs.mudynamics.com/2009/04/03/interactive-couchdb/ [Accessed: March 20, 2009].
4) RonDuPlain, "Apache-CouchDB Wiki", 16 May 2009 [Online]. Available: http://wiki.apache.org/couchdb/ [Accessed: May 20, 2009].
5) "Home Page - Amazon Web Services". 2009, [Online]. Available: http://aws.amazon.com/simpledb/ [Accessed: March 13, 2009].
6) James Murthy, "Programming Amazon Web Services", Available: http://books.google.co.in/books?id=xIauw5xWTO8C&pg=PA497&dq=simpledb&ei=OSxJSvSuG4j6lQTNgrnBAg [Accessed: March 28, 2009].
7) Tod Hoff ," The Search for the Source of Data - How SimpleDB Differs from a RDBMS", 22 April 2008 [Online]. Available: http://highscalability.com/search-source-data-how-simpledb-differs-rdbms [Accessed: March 20,2009].
8) Stefan Reuter, Amazon SimpleDB Performance", 28 August 2008 [Online]. Available: "http://blogs.reucon.com/srt/2008/05/08/amazon_simpledb_performance.html" [Accessed: May 2,2009].
9) Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, " Bigtable: A Distributed Storage System for Structured Data", November 2006 [Online]. Available: http://labs.google.com/papers/bigtable.html [Accessed: May 3, 2009].
10) Andrew Hitchcock, BigTable As A Web Service, [Online]. Available: http://bigtable.appspot.com/ [Accessed: May 12, 2009].
11) Alvanos Michalis, "Bigtable: A Distributed Storage System for Structured Data", May 8 2009 [Online]. Available: http://www.csd.uoc.gr/~hy558/reports/report_bigtable.pdf [Accessed: May 20,2009].