ABSTRACT
As scientific computing problems become more and more data intensive, many technologies and systems have been developed to efficiently store and serve terabytes or even petabytes of data. One example is the HBase system developed by Yahoo!, which is an open source implementation of the BigTable system originated from Google. HBase supports reliable storage and efficient access of billions of rows of semi-structured data. On the other hand, the Apache Cassandra developed by Facebook, which can be describe as BigTable data model running on an Amazon Dynamo-like infrastructure. Cassandra is designed to handle very large amount of data spread across many commodity servers while providing a highly available service with no single point of failure. The main objective of the project is to evaluate the performance of Hbase and compare it with Cassandra.

General Terms
Performance, Design, Timeline.

Keywords
HBase, Cassandra, HPC cluster, wordcount.

1. INTRODUCTION
The concept of NoSQL differs from the standard Relational Database. The problems of the relational databases included the inability to work on data-intensive applications and indexing of large number of files/documents. Many NoSQL systems have been developed in order to cater to the above requirements.

Many of the more popular NoSQL databases have of late been distributed in nature. This type of structure means redundant storage of data on many servers. The storing occurs using a distributed hash table. Generally in a distributed hash table, the data is stored and a keyspace is evaluated using a hash function. The hashing is done using a SHA-1 hash. The data is traversed and then stored in a node that is responsible for that keyspace. A keyspace partitioning scheme splits ownership of this keyspace among the participating nodes. An overlay network then connects the nodes, allowing them to find the owner of any given key in the keyspace.

Two very popular versions of NoSQL database using the concept of keyspace are Hbase and Apache Cassandra; the topics of our project.

2. TECHNOLOGY SURVEY

2.1 HBase
Hbase is an open-source, distributed, column-oriented sorted map data store modeled after Google’s BigTable. It runs on top of HDFS providing BigTable-like capabilities for Hadoop. It is useful when fault-tolerant, random, real-time read/write access to data stored in HDFS is required. HBase runs on top of HDFS.

Figure 1. An example of the BigTable data model []

HBase runs on top of HDFS and Figure 2 shows its architecture. Tables are horizontally split into regions, and regions are assigned to different region servers by the HBase master. Regions are further vertically divided into stores by column families, and stores are saved as store files in HDFS. Data replication in HDFS ensures high availability of HBase table data. During the runtime operations of the whole HBase system, the ZooKeeper is used to coordinate the activities of the master and region servers, and save a small amount of system metadata.

Figure 1. HBase architecture.

2.2 Cassandra
Cassandra is an open source distributed database management system. It is an Apache Software Foundation top-level project designed to handle very large amounts of data spread out across many commodity servers while providing a highly available service with no single point of failure. It is a NoSQL system that was initially developed by Facebook and it powers their Inbox Search feature. A standalone test version of Twitter called “Twissandra” has also been created as demonstration. The basic fundamental of Cassandra is that it is a columnar database or rather a column-oriented distributed database. The data is stored in the form of columns and it is uniquely marked using ‘keyspace’. It can be classified as a ‘Cloud Db’ similar to HBase.

For instance: usrs[’adwaraka’] will indicate a column family of users. In it, there will be an identifier “adwaraka”. In ‘usrs’, we can further add usrs[’adwaraka’][’fname’], usrs[’adwaraka’][’lname’] and usrs[’adwaraka’][’gender’].
2.2.1 ARCHITECTURE DESIGN

**Column and Column Family:** As mentioned before, the data model is columnar in nature. The column is the base of Cassandra data model. The column is the lowest and smallest increment of data. It’s a tuple (triplet) that contains a name, a value and a timestamp.

Here’s a column represented in JSON notation:

For the user [adwaraka]

```json
{
  fname: "Arvind",
  lname: "Dwarakanath",
  gender: "Male"
}
```

A column family resembles a table in an RDBMS. Column families contain rows and columns. Each row is uniquely identified by a row key. Each row has multiple columns, each of which has a name, value, and a timestamp. Unlike a table in an RDBMS, different rows in the same column family do not have to share the same set of columns, and a column may be added to one or multiple rows at any time. It can be useful to distinguish between “static” column families that contain values such as user data or other object data, and “dynamic” column families that contain data such as precalculated query results.

**Keyspaces:** Keyspaces group column families together. Typically, there will be one Keyspace for each application that uses a Cassandra cluster. The most important settings that are defined at the keyspace level are the replication factor and the replica placement strategy. Thus, if you have sets of data that have different requirements for these settings (such as different levels of fault-tolerance), these sets of data should reside in different keyspaces. A keyspace is to be set before any client API like thrift has to be fired. On the Cassandra CLI, use the ‘use <keyspace name>‘ to select the required keyspace. The command goes like this

```bash
use keyspace Keyspace1;
```

**Super Columns:** Super Columns are a type of super structure of columns. Super columns are way to group multiple columns. Every super column must have a different name, just like with regular columns. Different super columns may hold sub columns with the same name. Super columns are a way to add an extra map layer to the data model. Super columns are frequently used to hold a single record where each field in the record is represented by a sub column. For example, the name of a super column might be the ID of a transaction and each sub-column could hold some attribute of the transaction.

For example, if a transactions row like the one describe had two entries, it might look like:

```json
trans-A: {
  date: "01/02/2010",
  amount: 5000
  timespace: <value1> }
```

```json
trans-B: {
  date: "01/03/2010",
  amount: 4500
  timespace: <value2> }
```

**Major Client Libraries for Cassandra: Thrift**

Thrift is a software framework that allows for scalable cross-programming development. In this context, Thrift is the name of the RPC client used to communicate with the Cassandra server. It statically generates an interface for serialization in a variety of languages, including C++, Java, Python, PHP, Perl, C# to name a few. It is this mechanism that allows you to interact with Cassandra from any of these client languages.

Some other clients that are used include Hector (using Java), Pycassa (using Python), phpccasa (PHP), Ruby (Cassandra) etc. The libraries are available at github website.

3. ARCHITECTURE DESIGN

The main objective of the project is to evaluate the performance of Hbase and compare it with Cassandra; as mentioned in the abstract. To do so, we need to have a bench marking program. So the program selected for the same is the Word count Algorithm. The Word Count is a favorite of the Hadoop Introductory course and is synonymous with ‘Hello World’ of a language.

In general, Word Count parses the input files and returns essentially two values – the word that is counted and the number of times it has repeated itself.

We decided to use the Word Count and the output will be stored in columns in the Cloud Db and therefore used to compare the performances. We would compare the read and the write operations.

4. IMPLEMENTATION TIMELINE

- Get familiar with HBase and do a broad study of what HBase and Cassandra benchmarking techniques are. (1 week)
- Complete the setup for HBase and Cassandra (2 weeks).
- Do a read/write technique (3 weeks).
- Integrate additional components like Lucene Index to see the boost in performance.

5. REFERENCES


[2] _Hadoop Hbase-0.20.2 Performance Evaluation_ by Kareem Dana at Duke University. It shows a varied set of test cases for executions to test Hbase.