Performance Analysis of Cloud Databases Handling Social Networking Data

Niladri Sekhar Dey
Department of Information Technology
Padmasri Dr. B. V. Raju Institute of Technology
Hyderabad, India
niladri.dey@bvrit.ac.in

Dr. Sujoy Bhattacharya
Department of Information Technology
Padmasri Dr. B. V. Raju Institute of Technology
Hyderabad, India
sujoy.b@bvrit.ac.in

Abstract—With the growing popularity of Social Networking, the need for storage and analysis of data generated by such applications has increased in leaps and bounds. With the advent of cloud computing tools that handle large amounts of data with ease, there is and increased usage of such tools to manage social networking data. However, usage of tools that can be employed for accessing and analyzing social networking data needs to be optimized. Performance of such tools largely depends on the nature of the database that is used in the back-end. This research illustrates that the choice of database decides to a large extent the performance of the tool and related application. This research will also bring in clarity on using social networking tools for specific purposes and the effect of MapReduce on various storage structures.

Keywords—SNA(Social Networks Analysis), HBASE, SimpleDB, MapReduce, AWS, Crawler, Reader, FB API, Twitter API

I. INTRODUCTION

The analytical study of social networking data is very effective in order to achieve significant milestones in research of social structure, social behaviors, trends and characteristics of the framework [1][2][3]. The analysis can lead to multiple momentous decisions related to marketing, social studies and technology research. The investigation of social network data and then analyzing the same leads to major complications as this data is mostly oversized and unstructured. Moreover, the choice of appropriate applications is very important while analyzing the data. This problem is largely addressed using the cloud computing applications, which allow elasticity on infrastructure and unstructured storage of data [4]. As the data is the major source of analytical research, hence multiple migrations are connected to each analytical application and few social networking frameworks have their own preference on database types. Multiple cloud service providers provide multiple different databases, which have their own performance issues, benefits and policies. This research enables the understanding of performance comparison on SimpleDB (Amazon Web Services or AWS) [5] and HBASE (Apache). The rest of the paper is organized as follows. In Part – II, we have elaborated some effective results on social networking data analysis and their performance issues. In Part – III, we have listed an application architecture which is used in this research. In Part – IV, we have compared the results obtained from experiments on SimpleDB and HBASE. In Part – V, the effect of MapReduce on the used databases is illustrated. In Part – VI, we explain the tools and systems and in Part – VII we summarize and conclude our work with references to the earlier sections.

II. RELATED WORK

A. AWS Based Social Networking Applications

The integration of social networking applications with cloud computing is not a rare example. Predominantly, the applications use cloud to host social sites or create scalable applications for social networking sites. The social networking site Facebook uses Amazon Web Services to build scalable applications and hosted over AWS with Elastic Cloud 2 and SimpleDB. The performance of Facebook on some rich content types is discussed in section IV.

B. Apache Based Social Networking Applications

The social networking applications also integrate with open source like Apache Foundation products. Another social networking site Twitter uses Apache Foundation products to host and store social networking data. Twitter uses apache HBASE across Apache Hadoop clusters, where HBASE allows the social networking engineers from twitter to run their MapReduce programs on HBASE [6][7]. HBASE allows twitter to periodically update rows of data with the help of Hadoop Distributed File System or HDFS [8]. The performance of Twitter on some rich content types is discussed in section – IV.

III. PERFORMANCE EVALUATION APPLICATION

The social networking data analysis is developed to understand the relation between multiple participants in the social network called actors. The actors generate data during multiple operations they perform. This dataset is called the social network data. To test the performance of HBASE and SimpleDB, a simple application was built during this research work, which allows multiple operations on social networking data. The application maintains a mirrored copy of the data both in HBÂSE and SimpleDB [9] and performs all the operations alternatively on both. The complete application (Fig. 1) has many parts, which are explained below [10][11]. A
simple java framework is used, specially for the APIs for Facebook and Twitter. Note that a developer account is needed to access the Facebook and Twitter APIs.

![Application Components](image)

**Fig 1: Application Components for Performance evaluation**

A. FACEBOOK Interface

The component Facebook interface is actually an open source package for Facebook and Java integration called RestFB, which is a simple and flexible Facebook Graph API and Old REST API client RestFB is a simple and flexible Facebook Graph API and Old REST API client. The package contains a few useful classes like com.restfb.batch, com.restfb.exception and com.restfb.util. The connection to Facebook through Java code using REST API sets has a few steps –

- **Initialization** – This step ensures the connectivity between Java client and Facebook using the personal access token. A personal access token can be obtained by joining the Facebook developer group with a valid Facebook account.

- **Fetching Single Object** – This step defines the conversion of Facebook objects into Java objects and farther mapping of those objects to User and Page types respectively.

B. Crawler

The Crawler components is not a URL crawler, rather this is a Facebook Post crawler. The Batch API in the com.restfb package is used. The Batch API in great if we have to retrieve multiple posts at a time. This component also has to perform the following steps –

- **BatchRequestBuilder** - In this step the batch request is built with specific command operation or queries.

- **BatchResponseBuilder** - In this step the response containers are initialized, so that the converted Facebook objects can be stored.

C. Contexter

The contexter component is basically a normalization component in this application. This component performs a few specific tasks in a specific order like Language detection, Named entity recognition, Anaphoric normalization and Text segmentation.

It was assumed that the common text will appear in English and the rest of the process will start with this consideration. The first step is to compute each component block to extract the text by applying the formulas

\[
P(X) = \frac{P}{D(Tx)}
\]

\[
D(Tx) = \sum_{t=v}^{w} D_t
\]

\[
P_{(x)_{i,j}} = T_{(x)_{j,d}}
\]

\[
P_{(x)_{i,j}} = T_{(x)_{i+1,j-1}}
\]

Where, \( P(X) \) is extracted text component

\( P \) is the total text block

\( D(Tx) \) is the domain of recognizable keywords

\( P_{(x)_{i,j}} \) is the extracted text component before mapping

\( T_{(x)_{i,j}} \) is the extracted text component after mapping

The named entry recognition algorithm is to find multiple small normalization-able components of texts (Eq. 1), where the domains of the known keywords are made from each collected keywords (Eq. 2). When the final text is extracted, (which can actually be many pieces of text), the mapping process starts. This mapping process eventually normalizes the unstructured text. The mapping process maps the extracted texts to mapping fields (Eq. 3). Sometimes based on few extracted text, new fields also need to be created (Eq. 4).

D. Twitter Interface

This interface built on the Twitter4J is an unofficial Java library for the Twitter API. With Twitter4J, the integration of this application with the Twitter service is very effective because of the available classes.

E. Reader

This interface sets a timer, either a 90 second TCP level socket timeout, or a 90 second application level timer on the receipt of new data. If 90 seconds pass with no data received, including newlines, the system will disconnect and reconnect immediately. If the data is received successfully, then the Contexter component works again.
F. Analysis

This component interfaces between the query GUI and the database. This module also takes care of the pre-defined queries, result sets, user data display and finally the performance monitoring, which is elaborated in Section - IV.

IV. CLOUD DATABASE PERFORMANCE

The Cloud Databases like HBASE and SimpleDB are connected to this application, where the application sends all the queries to both the databases. The setup of the experiment is on a single node, the details of which are explained in part – VII. The basic and initial dataset was 10,000 rows with 10 fields in each, measuring about 10KB of data for each block. Hence the total size of the data used for this experiment was nearly 1GB. The application was also equipped with multi-threading of queries. Hence we also included an incremental type of multi-threading code, where initially the code will batch only 5 queries per second and then for each second 2 new threads will be generated. Hence after running the application for 1 hour, total number of queries will be approximately 7000 with a internet connection of 10 MBPS. With this setup, we recorded the response time for both HBASE (Local) and SimpleDB (AWS). The results are listed in Table 1 and Table 2 below:

<table>
<thead>
<tr>
<th>HBAS E</th>
<th>Query Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 MB</td>
<td>2 1.4 K 2.0 K 3.0 K 4.0 K</td>
</tr>
<tr>
<td>5 MB</td>
<td>2 6.98 K 9.58 K 15.6 K 16.3 K</td>
</tr>
<tr>
<td>10 MB</td>
<td>2 12 K 18.4 K 32.4 K 44.5 K</td>
</tr>
<tr>
<td>15 MB</td>
<td>2 20.88 K 28.46 K 46.2 K 66.9 K</td>
</tr>
</tbody>
</table>

All the time values are in ms, where k denotes thousand

<table>
<thead>
<tr>
<th>HBAS E</th>
<th>Query Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 MB</td>
<td>20 2.8 K 40 K 160 K 640 K</td>
</tr>
<tr>
<td>5 MB</td>
<td>20 14 K 56 K 224 K 896 K</td>
</tr>
<tr>
<td>10 MB</td>
<td>20 28 K 112 K 448 K 1702 K</td>
</tr>
<tr>
<td>15 MB</td>
<td>20 42 K 168 K 672 K 1952 K</td>
</tr>
</tbody>
</table>

All the time values are in ms, where k denotes thousand

The results are compared in a graphical format in Fig 2 and Fig 3 below:

![Fig 2: Query Vs Data Set Size in HBASE](image)

![Fig 3: Query Vs Data Set Size in SimpleDB](image)

We have noticed a significant difference in the response time between SimpleDB (Fig. 2) and HBASE (Fig. 3), where HBASE applications are deployed locally on the Apache Tomcat server and SimpleDB applications are deployed on Elastic Cloud Server on AWS.

\[
Q_R = DB_s * Q_N * 2 + N_s, \quad N_s = 0
\]

\[
Q_R = DB_s * Q_N * 4 + N_s, \quad N_s = 10
\]

Where, \(Q_R\) is Query Response Time, \(DB_s\) is size of the database domain to scan, \(Q_N\) is number of Queries to be fired in a second, \(N_s\) is the network speed to be considered at 0 for local

We have listed the formulas for calculating the query response times for SimpleDB and HBASE, where the number of queries, amount of data and network speed have to be considered. Here after analysing the result sets we found the formula for calculating the Response Time for HBASE (Eq. 5) and SimpleDB (Eq. 6).

V. EFFECT OF MAPREDUCE ON CLOUD DATABASES

A generic MapReduce is a software programming model for compiling any large data sets for any distributed and parallel algorithms on a configured cluster. Any simple
MapReduce program consists of a Map() procedure, which performs filtering and sorting and a Reduce() procedure that performs a summarization operation. We used a MapReduce library for HBASE called Amazon Hadoop MapReduce and one for SimpleDB called Amazon Elastic Cloud MapReduce. After applying the MapReduce algorithms on HBASE and SimpleDB we noticed an astounding 15% performance improvement in both the cases, irrespective of network speed as they can be configured locally on Virtual Machine or distributed clusters. We used the following standard algorithm of MapReduce [12]:

- Prepare the Map() input – The "MapReduce system" allocates Map processors, assigns the K1 input key value each processor would work on, and provides that processor with all the input data associated with that key value.
- Run the user-provided Map() code – Map() is run exactly once for each K1 key value, generating output organized by key values K2.
- “Shuffle” the Map output to the Reduce processors – the MapReduce system designates Reduce processors, assigns the K2 key value each processor would work on, and provides that processor with all the Map-generated data associated with that key value.
- Run the user-provided Reduce() code – Reduce() is run exactly once for each K2 key value produced by the Map step.
- Produce the final output – the MapReduce system collects all the Reduce output, and sorts it by K2 to produce the final outcome.

VI. CONCLUSION

The performance of HBASE and SimpleDB is analyzed on a social network dataset, which is large in volume and the effect of the number of queries on the same dataset is studied. It is proven that there is a large difference in terms of query response time depending on the database used. We have also noticed that the query response time is highly dependent on the network speed or the network bandwidth, through which the AWS is accessed. But in both the cases we found that the effect of MapReduce is significant. There is a large effect in case of a local VM that clustered or shared clustered and for Amazon Elastic Cloud MapReduce as well. During the Crawler and Reader implementation we found that the normalization process is easier for Facebook data for SimpleDB and Twitter data for HBASE. The conclusion is that due to internal use of the same databases the social data from both the social networking sites are compatible with SimpleDB and HBASE.

ACKNOWLEDGMENT

The research work is being carried in the Cloud Computing Center in Padmasri Dr. B. V. Raju Institute of Technology, Hyderabad. We would like to thank the management for allocating necessary funds for the research.

REFERENCES


