IITB Summer Internship 2014

Project Report

Load Testing and Benchmarking for BigData

Principal Investigator
Prof. D.B. Phatak

Project In-Charge
Mr. Nagesh Karmali

Project Mentors	Project Team Members
	Jayam Modi
Mr. Pqr	Sunil Raiyani
	Aayush Agrawal

July 2, 2014

Summer Internship 2014
Project Approval Certificate

Department of Computer Science and Engineering

Indian Institute of Technology Bombay

The project entitled “Load Testing and Benchmarking for Bigdata” submitted by Jayam Modi, Sunil Raiyani, Aayush Agrawal, is approved for Summer Internship 2014 programme from 10th May 2014 to 6th July 2014, at Department of Computer Science and Engineering, IIT Bombay.

Prof. Deepak B. Phatak	Mr. Nagesh Karmali
Dept of CSE, IITB	Dept of CSE, IITB
Principal Investigator	Project In-charge

Mr. Pqr
Dept of CSE, IITB
External Examiner

Place: IIT Bombay, Mumbai
Date: July 2, 2014

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/-fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

____
Jayam Modi
Sardar Vallabhbhai National Institute of Technology, Surat

____
Sunil Raiyani
Sardar Vallabhbhai National Institute of Technology, Surat

____
Aayush Agrawal
Maulana Azad National Institute of Technology, Bhopal

Date: ______________________

Abstract

Write your abstract here

List of Figures

List of Tables

Contents

Chapter 1

Introduction to BigData and Benchmarking

1.1 Introduction

The project is about Load Testing and Benchmarking for BigData. A distributed file system setup is generated using a cluster consisting of multiple nodes. Complex data sets are generated and the data is processed using distributed processing tools to produce meaningful results.

1.2 Load Testing

It means to test the system by steadily increasing the load on the system till it reaches its threshold limit. In the current case, the size of the dataset is increased until it becomes impossible for the cluster to process the data. It helps to identify the maximum operating capacity of the system and the bottlenecks if any. The components causing degradation are easily identified using load testing.

1.3 Benchmarking

It refers to the process of comparing the performance metrics of own systems with the industry standards. It is performed using a specific indicator which becomes a performance metric for comparision. Its aim is to help evolve systems in those areas where they are weak in performance. Benchmarking software can be used to organize huge and complex information.

1.4 BigData

It is a term that covers data sets so large and complex that it becomes impossible to process them using on-hand database management tools and traditional data processing applications. Relational database management systems fail to perform when it comes to BigData. It largely involves unstructured data which is not possible to capture and process using DBMS or RDBMS. The only possible way to process BigData is using parallel and distributed database systems. BigData sets are so large that their size is measured in terms of exabytes (2*10^{18} bytes). It is currently impossible for any single system to store and process such a huge amount of data on its own.

1.5 Hadoop

Apache Hadoop is an open-source framework that can be used for storing and processing large and complex data sets on clusters made up of commodity hardware systems. It provides a Distributed file system named HDFS which is a platform to store huge amounts of data divided into blocks across multiple hosts. It also provides the MapReduce Engine which performs the processing of BigData.

Chapter 2

Apache Hadoop

2.1 Introduction

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. The project includes these modules:

- Hadoop Common
- Hadoop Distributed File System
- Hadoop Yarn
- Hadoop MapReduce

2.1.1 Hadoop Common

Hadoop Common is the set of common utilities that support other Hadoop modules. In this section :

- Using File System(FS) shell commands: The File System (FS) shell includes various shell-like commands that directly interact with the Hadoop Distributed File System (HDFS)
  Various commands are:

  1. cat: Copies source paths to stdout.
     hdfs dfs -cat file:///file3 /user/hadoop/file4
  2. chmod: Change the permissions of files. With -R, make the change recursively through the directory structure. The user must be the owner of the file, or else a super-user.
  3. chown: Change the owner of files. With -R, make the change recursively through the directory structure. The user must be a super-user.
  4. copyFromLocal: Copy single src, or multiple srcs from local file system to the destination file system.
     Usage: hdfs dfs -copyFromLocal <localsrc> URI

5. copyToLocal:Copy files to the local file system.
   Usage: hdfs dfs -copyToLocal [-ignorecrc] [-crc] URI <localdst>

- Hadoop Commands References: All hadoop commands are invoked by the bin/hadoop script. Running the hadoop script without any arguments prints the description for all commands.

  1. fsck: Runs a HDFS filesystem checking utility.It is used to find out which files and blocks are corrupt.
  2. jar: Runs a jar file. Users can bundle their Map Reduce code in a jar file and execute it using this command.
     Usage: hadoop jar <jar> [mainClass] args...
  3. version: Prints the current version.
  4. dfsadmin: Runs a HDFS dfsadmin client.
     Usage: hadoop dfsadmin -report: Reports basic filesystem information and statistics.

2.1.2 Hadoop Distributed File System

HDFS is a distributed file system that provides high-throughput access to data. It provides a limited interface for managing the file system to allow it to scale and provide high throughput. HDFS creates multiple replicas of each data block and distributes them on computers throughout a cluster to enable reliable and rapid access.[1]

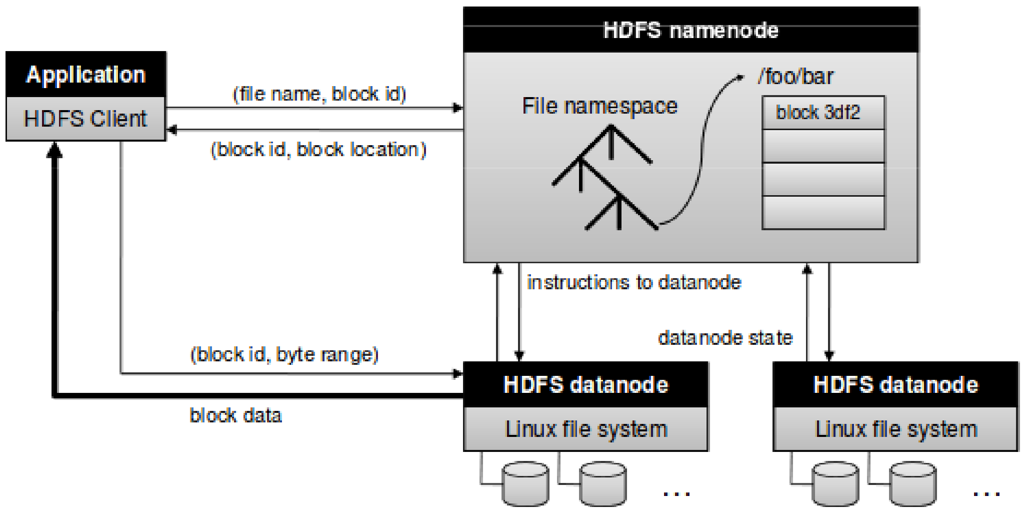


Figure 2.1: Hadoop Architecture

1. Namenode and Datanode: The distributed file system adopts a master slave architecture in which the namenode maintains the file namespace (metadata, directory structure, file to block mapping, location of blocks, and access permissions) and the datanodes manage the actual data blocks.

2. Relationship between Namenode and Datanode: Data nodes continuously loop, asking the name node for instructions by sending heartbeat messages. A name node can't connect directly to a data node; it simply returns values from functions invoked by a data node. Each data node maintains an open server socket so that client code or other data nodes can read or write data.

3. Data Replication: HDFS replicates file blocks for fault tolerance. An application can specify the number of replicas of a file at the time it is created, and this number can be changed any time after that. The name node makes all decisions concerning block replication.The namenode attempts to optimize communications between data nodes. The namenode identifies the location of data nodes by their rack IDs.

4. Data Organization: Hadoop primary goal is to store large datafiles.The default size of typical datablock is 64MB.It can be configured by changing the core-site.xml file. HDFS tries to place each block on separate data nodes.

5. Data Block Rebalancing: HDFS data blocks might not always be placed uniformly across data nodes, meaning that the used space for one or more data nodes can be underutilized. It provides hadoop balance command for manually rebalancing task.

6. Snapshots: HDFS was originally planned to support snapshots that can be used to roll back a corrupted HDFS instance to a previous state.

2.1.3 Hadoop Yarn

The fundamental idea of YARN is to split up the two major responsibilities of the JobTracker i.e.resource management and job scheduling/monitoring, into separate daemons:a global ResourceManager and per-application ApplicationMaster (AM)
The ResourceManager and per-node slave, the NodeManager (NM), form the new, and generic, system for managing applications in a distributed manner.
The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system.
The Hadoop Yarn architecture can be seen here 2.2

2.1.4 Hadoop MapReduce

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.[2] The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

The MapReduce framework operates exclusively on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, mainly of different types.
MapReduce-Interfaces

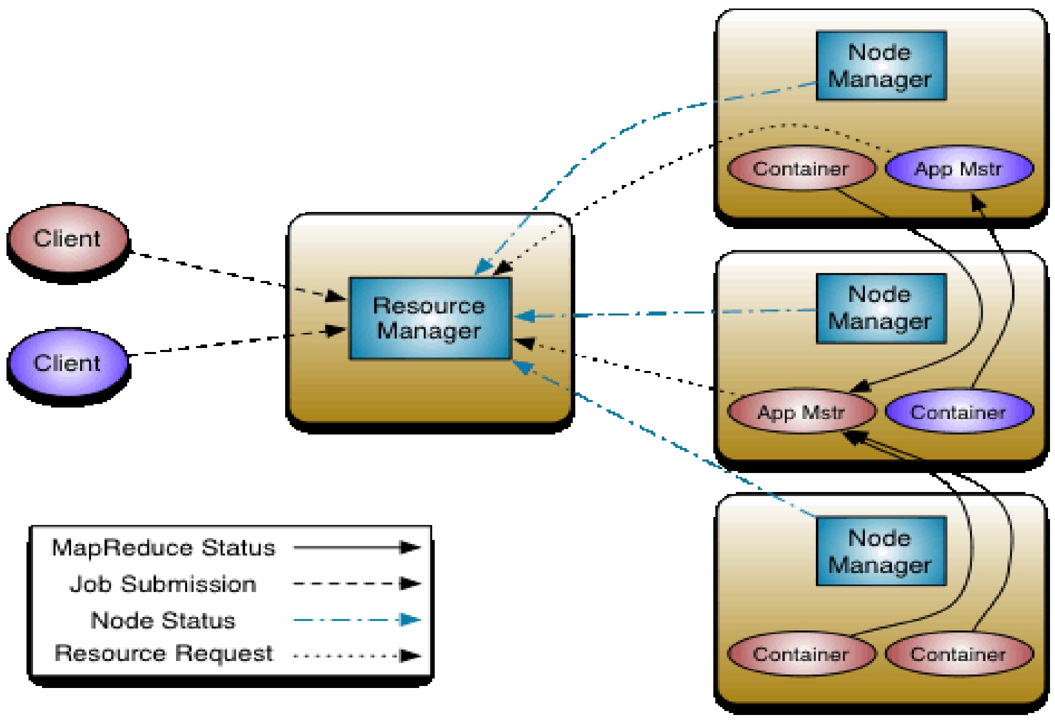


Figure 2.2: Yarn Architecture

- Mapper: Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs.
- Reducer: It reduces a set of intermediate values which share a key to a smaller set of values
- Partitioner: It controls the partitioning of the keys of the intermediate map-outputs. The key (or a subset of the key) is used to derive the partition, typically by a hash function.
- Combiner: They are an optimization in MapReduce that allow for local[3] aggregation.Combiners works as mini-reducers that take place on the output of the mappers.2.3

2.2 Installation

The main goal of this tutorial is to get a simple Hadoop installation up and running so that we could learn more internals about it. This tutorial has been tested on [4]

1. Ubuntu Linux 14.04 LTS
2. Hadoop 2.2 released on October 2013

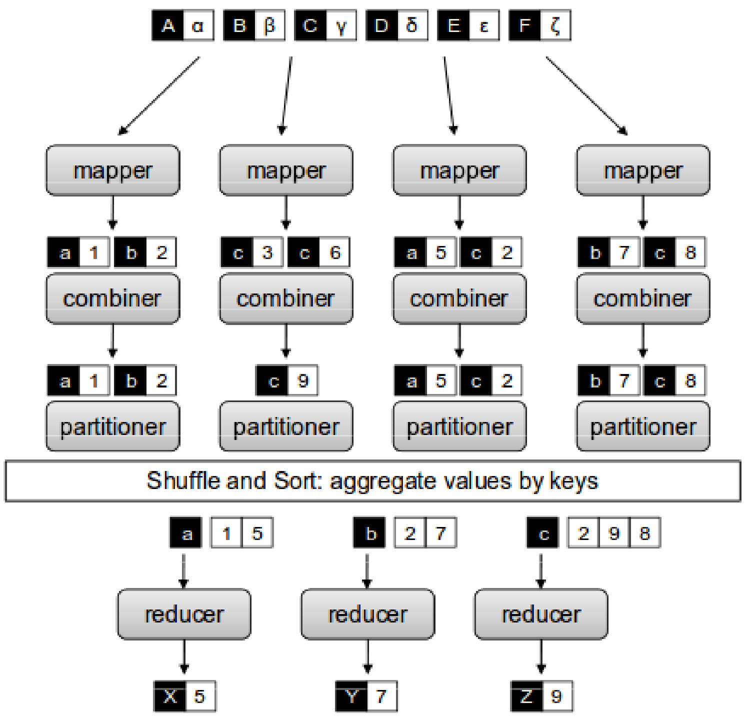


Figure 2.3: Complete View of MapReduce

2.2.1 Single Node Cluster

- Prerequisites

  1. Hadoop Client/User
     ```
     sudo addgroup hadoop
     sudo adduser ingroup hadoop hduser ## Name assigned to the client groups
     sudo adduser hduser sudo
     ```

  2. Java jdk(6 or higher)
     ```
     sudo apt-get install openjdk-7-jdk
     cd /usr/lib/jvm
     ln -s java-7-openjdk-amd64 jdk
     ```

  3. SSH
     ```
     sudo apt-get install openssh-server
     ssh-keygen -t rsa -P
     ssh localhost
     ```

- Installing Hadoop-Download and Environment:

  1. Download
     ```
     wget http://apache.mirrors.lucidnetworks.net/hadoop/common/stable/hadoop- 2.2.0.tar.gz
     ```

```
sudo tar vxzf hadoop-2.2.0.tar.gz -C /usr/local
cd /usr/local
sudo mv hadoop-2.2.0 hadoop
sudo chown -R hduser:hadoop hadoop
```

2. Enviornment Set-up :Add following to .bashrc file
   ```
   export JAVA_HOME =/usr/lib/jvm/jdk/
   export HADOOP_INSTALL=/usr/local/hadoop
   export PATH=PATH :HADOOP_INSTALL/bin
   sudo mv hadoop-2.2.0 hadoop
   sudo chown -R hduser:hadoop hadoop
   ```

- Installing Hadoop-Configuration and Launching:

  1. User based configuration: change the extension of configuration file
     ```
     /usr/local/hadoop/etc/hadoop/core-site.xml
     /usr/local/hadoop/etc/hadoop/hdfs-site.xml
     /usr/local/hadoop/etc/hadoop/yarn-site.xml
     /usr/local/hadoop/etc/hadoop/mapred-site.xml
     ```

  2. Launch
     ```
     mkdir -p mydata/hdfs/namenode
     mkdir -p mydata/hdfs/datanode
     hdfs namenode -format
     start-dfs.sh
     start-yarn.sh
     ```

2.2.2 Multi Node Cluster

- Prerequisites Install the single node cluster on every node and try to run some task on these nodes.[5]

- Network Settings To run a nulti-node cluster ensure that master and slave are on the same network.Identify the ip address of each node. Now make entries in the /etc/hosts file as follows:
  10.129.46.111 master
  10.129.46.113 slave

- SSH Access Add the public key of master to all the slaves using the command:
  ```
  hduser@master: $ ssh-copy-id -i $HOME/.ssh/id_rsa.pub hduser@slave
  ```

  Now ssh to master and slaves ensuring the passwordless access.

```
ssh master

ssh slave
```

- Configuration files Add the following lines between <configuration> ans </configuration> tags to the file in $HADOOP_HOME/etc/hadoop for both master and slave.

  1. Core-xite.xml
     ```
     <property >
     <name>fs.defaultFS </name>
     <value>hdfs://master:9000 </value>
     </property>
     ```

  2. yarn-site.xml
     ```
     <property >
     <name>yarn.nodemanager.aux-services</name>
     <value>mapreduce_shuffle </value>
     </property>
     <property >
     <name>yarn.nodemanager.aux-services.mapreduce_shuffle.class </name>
     <value>org.apache.hadoop.mapred.ShuffleHandler </value>
     </property>
     ```

  3. mapred-site.xml
     ```
     <property >
     <name>mapreduce.framework.name</name>
     <value>yarn</value>
     </property>
     ```

  4. hdfs-site.xml
     ```
     <property >
     <name>dfs.replication</name>
     <value>3</value>
     </property>
     <property >
     <name>dfs.namenode.name.dir</name>
     <value>file:/home/hduser/mydata/hdfs/namenode</value>
     </property>
     <property >
     <name>dfs.namenode.data.dir</name>
     <value>file:/home/hduser/mydata/hdfs/datanode</value>
     </property>
     ```

Now add all the slaves name to the $HADOOP_HOME/etc/slaves file

```
nano $HADOOP_HOME/etc/slaves
```

- Launching Run the following script in master node

  ```
  hdfs namenode -format
  start-dfs.sh
  start-yarn.sh
  ```

  To test whether all the daemon started or not run the jps command on master and slave.

  On Master

  ```
  hduser@master:/usr.local/hadoop$:  jps
  9412 SecondaryNameNode
  9834 NameNode
  1056 jps
  10898 ResourceManager
  ```

  On Slave

  ```
  hduser@slave:/usr.local/hadoop$:  jps
  1876 NodeManager
  9856 DataNode
  10561 jps
  ```

2.3 Network Monitoring Tools

Network monitoring is the use of a system that constantly monitors a computer network or cluster for slow or failing components and that notifies the network master (via graphs).There are many networking tools such as Nagios,Ambari etc. Here ,we have used the ganglia as a monitoring tool.

2.3.1 Ganglia

Ganglia is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It consists of two daemon:

- gmond (Ganglia Monitoring Daemon)
- gmetad (Ganglia Meta Daemon)

gmond runs on each node you want to monitor. It monitors changes in the host state, announce relevant changes, listen to the state of all other ganglia nodes via a unicast or multicast channel and answers requests for an XML.
gmetad runs on the master node and gathers all information from the client nodes. Ganglia also contains a PHP Web Front-end which displays the gathered information in the form of graphs via web pages.

1. **Installation on Master Node** The gmond daemon has to be installed on all nodes while the gmetad daemone must only be installed on the master node. To install ganglia and its web-frontend on the Master Node, fire the following commands from the terminal.[6]

   ```
   sudo apt-get install ganglia-monitor rrdtool gmetad ganglia-webfrontend
   ```

   Now edit the /etc/ganglia/gmetad.conf

   ```
   sudo nano /etc/ganglia/gmetad.conf
   ```

   Find the line of the following type and modify it as follows:

   ```
   data_source "master" 50 127.0.0.1 ip-address-of-namenode
   ```

   Here master is the name of cluster. 50 indicates that logs will be collected after every 50 seconds. 127.0.0.1 is ip address of a namenode.
   Now edit the file /etc/ganglia/gmond.conf

   ```
   sudo nano /etc/ganglia/gmond.conf
   ```

   Find the following lines and make appropriate changes as indicated:

   ```
   cluster {
   name = "master" ## Name assigned to the client groups
   owner = "unspecified"
   latlong = "unspecified"
   url = "unspecified"


   udp_send_channel {
   #mcast_join=239.2.11.71
   host = 10.105.24.11
   port = 8649
   ttl = 1


   udp_recv_channel {
   port = 8649

   tcp_accept_channel {
   port = 8649
   ```

   The changes in the above configuration file show that the master node which has IP address 127.0.0.1 will collect data from all nodes on tcp and udp port 8649. Now start the services using the following commands:

   ```
   sudo /etc/init.d/ganglia-monitor restart

   sudo /etc/init.d/gmetad restart
   ```

2. **Installation on Slave Node** Install the ganglia monitor package for all the slave nodes that we want to monitor.

   ```
   sudo apt-get install ganglia-monitor
   ```

Now edit the /etc/ganglia/gmond.conf file as follows:

```
sudo nano /etc/ganglia/gmond.conf
```

Make the following changes:

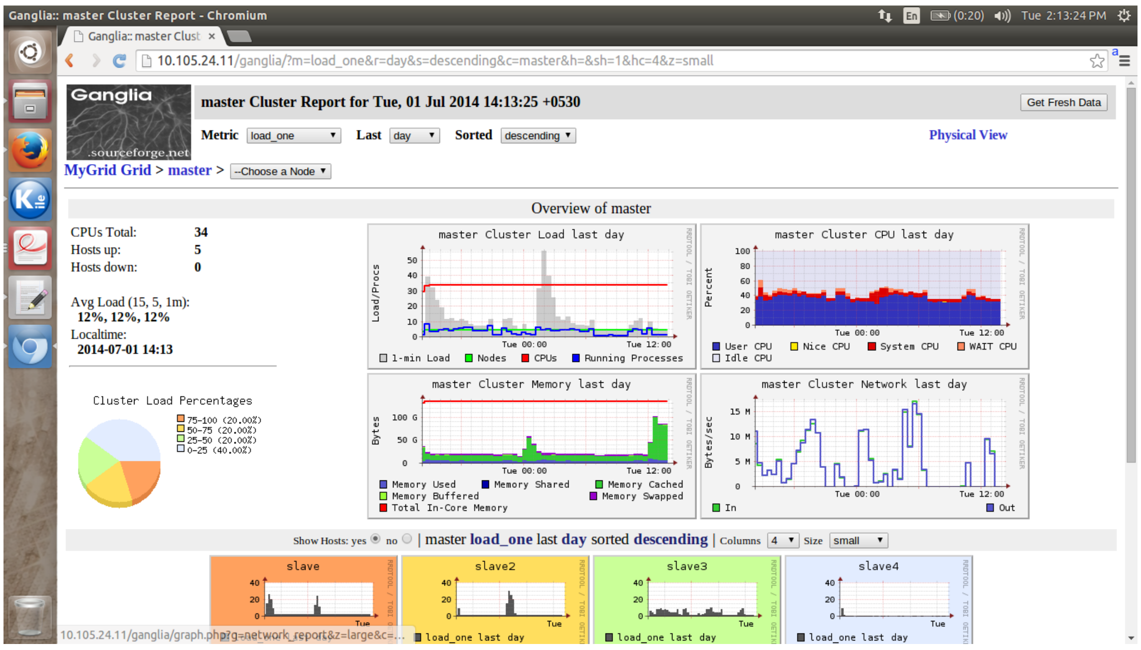
```
cluster {
name = "master" ## Name assigned to the client groups
owner = "unspecified"
latlong = "unspecified"
url = "unspecified"


udp_send_channel {
#mcast_join=239.2.11.71
host = 10.105.24.11
port = 8649
ttl = 1
```

Now restart ganglia-monitor service.

```
sudo /etc/init.d/ganglia-monitor restart
```

3. **Using Ganglia** Start a web browser and type in the following address : **http://ip-address-of-namenode/ganglia**
   A screen similar to the one shown in the figure 2.4 appears: Select the node that



Ganglia:: master Cluster Report - Chromium

10.105.24.11/ganglia/?m=load_one&r=day&s=descending&c=master&h=&sh=1&hc=4&z=small

master Cluster Report for Tue, 01 Jul 2014 14:13:25 +0530

Metric load_one Last day Sorted descending

MyGrid Grid > master > --Choose a Node

Overview of master

CPUs Total: 34
Hosts up: 5
Hosts down: 0

Avg Load (15, 5, 1m):
12%, 12%, 12%
Localtime:
2014-07-01 14:13

Show Hosts: yes [✓] no | master load_one last day sorted descending | Columns 4 Size small

10.105.24.11/ganglia/graph.php?g=network_report&z=large&c=...

Figure 2.4: Ganglia Interface

you want to monitor from the list of available nodes.2.4 There are many graphs for each node which indicate the Memory Usage, CPU utilization, Network Load and Load Sharing of various nodes.

Chapter 3

Apache Hive

3.1 Overview

Apache Hive, as reported in [7] , is a data warehouse infrastructure built on top of hadoop for providing data analysis and querying features. It was initially developed by Facebook but now it is used by many other companies. It supports HiveQL which is a SQL-like declarative language. The queries of HiveQL are compiled into map-reduce jobs executed on Hadoop. It also supports custom map-reduce scripts which can be inserted into queries.

3.2 Data Model

The Data in hive is organized into Tables, Partitions and Buckets.

- **Tables** - They are similar to tables from RDBMS. Each table is stored on a separate directory on the HDFS. The data is serialized and then stored on the files in the directory.

- **Partitions** - Each table can have multiple partitions which are stored in different subdirectories within the parent directory of the table.

- **Buckets** - The data of each partition may be divided into buckets depending on the hash of a column in the table. All the buckets are stored in different files within the partition sub-directory.

Hive provides primitive(integer, float, string,etc.) as well as custom data types (array and map).

3.3 Query Language

HiveQL supports all types of operations like select, project, aggregate, join, union, etc. It provides DDL statements to create, modify and delete tables. It also provides DML statements like load and insert to enter data into tables. HiveQL also supports multi-table insert which allows users to perform multiple queries on a single input data.

3.4 Installation

The following steps must be followed inorder to install hive on a system:

1. **Prerequisites**
   Setup Hadoop-2.2.0 using the steps mentioned in the previous chapter.

2. **Download Hive**
   Download the lastest version of hive from apache-hive's repository.
   ```
   wget http://apache.mirrors.hoobly.com/hive/stable/apache-hive-0.13.0-bin.tar.gz
   ```

3. **Extract Hive**
   Extract the files to a directory and then move the directory to a proper location.
   ```
   sudo tar -zxvf apache-hive-0.13.0-bin.tar.gz
   sudo mv apache-hive-0.13.0-bin /usr/local/hive
   sudo chown -R hduser:hadoop hive
   ```

4. **Setup Environment Variables**
   Add the following lines to /.bashrc file
   ```
   export HIVE_PREFIX=/usr/local/hive
   export PATH=$PATH:$HIVE_PREFIX/bin
   ```
   Now logout and then login again inorder to set the enviroment variables.

5. **Using Hive**
   Write **hive** on the terminal in order to open hive. Once opened, it will look like this:
   ```
   hive>
   ```
   Now, HiveQL statements can be used to analyze and manipulate data.

Chapter 4

BigBench

4.1 Overview

BigBench is an industry standard benchmark for big data analytics [8]. All the major characteristics in the lifecycle of a big data system are covered in BigBench which is an end-to-end benchmark. The three V's described by Douglas Laney [9] are the most important characteristics of a big data system:

- **Volume** - Large data set sizes.
- **Velocity** - Higher data arrival rates such as clickstreams.
- **Variety** - Different data type such as structured (relational tables) , semi-structured (key-value web-clicks) and unstructured (social media content).

4.2 Data Model

BigBench is based on a data model of a fictitious retailer who sells products to customers via physical and online stores. It has been adapted from the TPC-DS data model for relational databases. The following diagram 4.1 explains the data model:

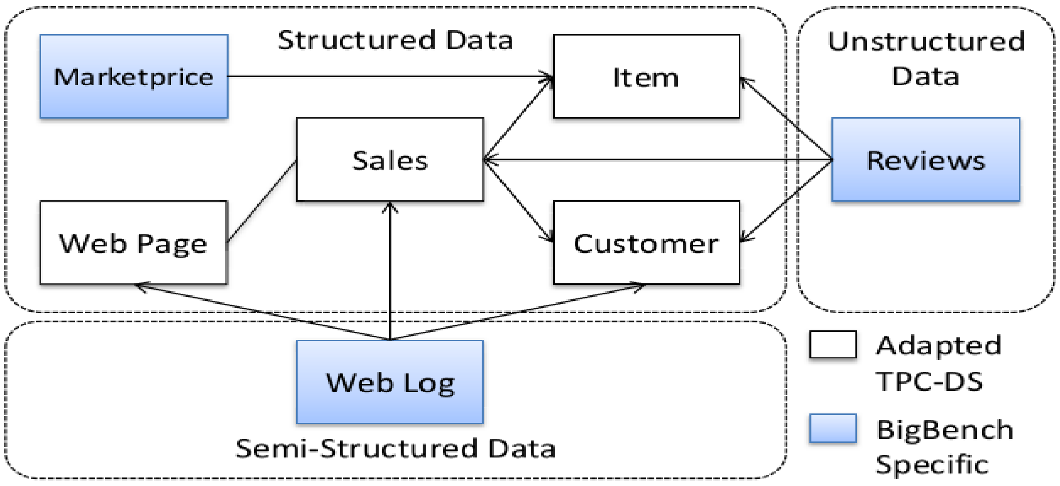


Figure 4.1: Big-Bench Data Model

This model is implemented using a set of 23 tables which contains various columns for storing structured data.

The Big-Bench Data model contains the following three types of data -

- **Structured** - The structured part of BigBench is an adaption of the TPC-DS model which also depicts a product retailer model. It borrows the store and online sales part from that model and adds a table named "MarketPrice" for competitor prices of the retailer.
- **Semi-Structured** - The semi-structured part's content is composed by clicks made by customers and guest users visiting the retailer site. Some of these clicks are for completing a customer order. The design assumes the semi-structured data to be a key-value format similar to Apache web server log format.
- **Un-Structured** - Online Product Reviews serve as a good source of unstructured data.

4.3 Data Generation

The Big-Bench data generation scheme is based on a technique called PDGF (Parallel Data Generation Framework). PDGF addresses only structured data by design. But it has been extended to generate semi-structured and unstructured data. PDGF is implemented in Java and is fully platform independent. The information for data generation is specified in two XML files, schema configuration (contains data similar to the relational schema) and generation configuration (contains additional post processing options).
The listing below shows the XML code for defining structured data.

```
<property name = "Item_marketprice" type="double" >
${item }*${avg_competitors_per_item}
</property>
<table name = "Item_marketprice">
<size> ${Item_marketprice} </size>
<field name = " imp_sk " size = " " type ="NUMERIC">
<gen_IdGenerator/>
</field>
 [..]
<field name = "imp_competitor" size ="20"
type = " VARCHAR " >
<gen_NullGenerator>
<probability> 0.00025 </probability>
<gen_RandomAString>
<size > 20 </size>
</gen_RandomAString>
</gen_NullGenerator>
</field>
 [..]
</table>
```

To generate a realistic web-log, all the required coloumns for a web-log entry are specified in a table. The sizing is computed based on a specific formula. The listing below shows the formatting code for the web-log. Some of the values are static while others are extracted from the table.

```
<output name = "CompiledTemplateOutput">
<template>
<!--
String nl = pdgf.util.Constants.DEFAULT_LINESEPARATOR;
buffer.append("127.0.0.1 - -["+fields[4]+":"
+fields[5]+"+0200]");
buffer.append("\"GET /page"+fields[7]+".html?");
[..]
buffer.append(" HTTP /1.1\" 200 0 -\" "+fields [1]) ;
buffer.append(" \" \" Mozilla /5.0 \" " + nl) ;
-->
</template>
</output>
```

The review generator for unstructured data was built as a standalone program which is configured using an XML document that specifies the parameters for each review. In order to generate correlated reviews, PDGF is used to generate the XML document for each review. The figure 4.2 shows the process of review generation. The process can

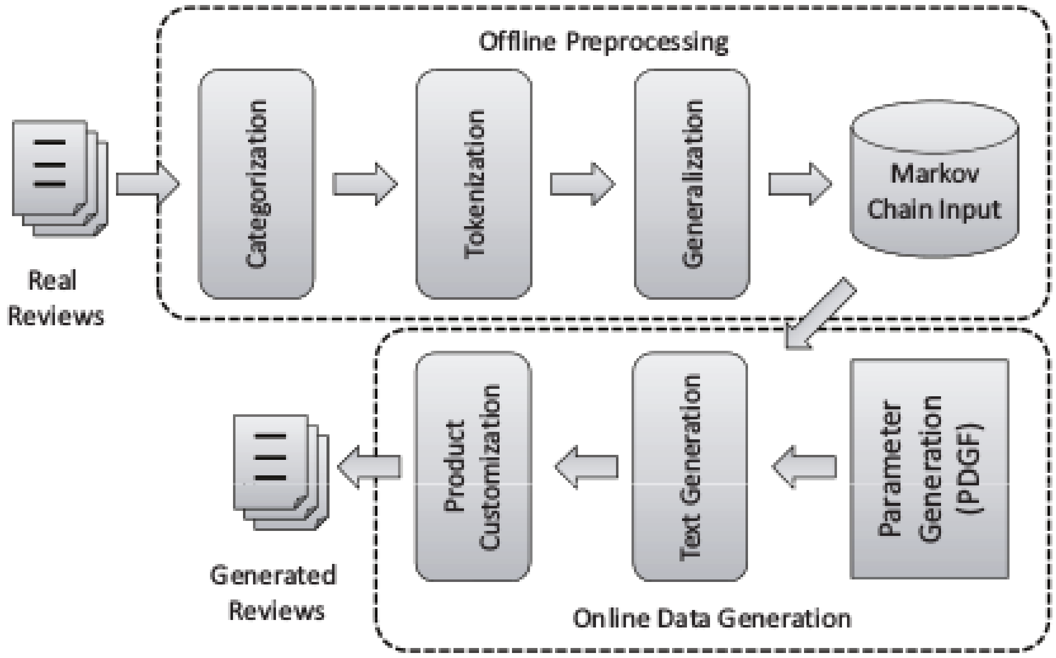
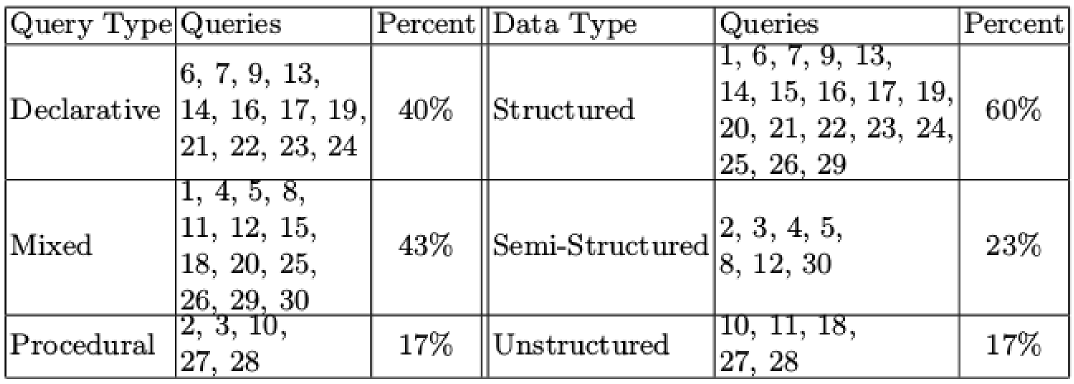


Figure 4.2: Review Generation Model

be separated in two phases. An offline phase, that processes real reviews and generates a knowledge base for the review generation and an online phase that generates reviews based on the knowledge base.

4.4 Workload

The workload for BigBench considers the initial database population in addition to the queries. This initial phase called Transformation Ingest (TI) covers all the steps needed to prepare the data before querying. The main part of the workload however is the set of 30 queries which are executed against the data model. They are designed along one business dimension and three technical dimensions. The set of 30 queries is chosen so that the distribution shown in figure 4.3 can be obtained. The figure clearly



Query Type	Queries	Percent	Data Type	Queries	Percent
Declarative	6, 7, 9, 13, 14, 16, 17, 19, 21, 22, 23, 24	40%	Structured	1, 6, 7, 9, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 29	60%
Mixed	1, 4, 5, 8, 11, 12, 15, 18, 20, 25, 26, 29, 30	43%	Semi-Structured	2, 3, 4, 5, 8, 12, 30	23%
Procedural	2, 3, 10, 27, 28	17%	Unstructured	10, 11, 18, 27, 28	17%

Figure 4.3: Workload Distribution

indicates that a major proportion of the queries operate on structured data since the data model largely consists of structured data.

4.5 Benchmark Results

The set of 30 queries was executed on a 8 node Teradata Aster appliance. Each node was a Dell server with two quad-core Xeon 5500 @ 3.07Ghz and hardware RAID 1 with 8 2.5“ drives. The table 4.1 shows the times obtained for individual queries.

Table 4.1: Query Run-Time

Query	run-time(sec)	Query	run-time(sec)
A1	200	A16	8700.045
A2	12.529	A17	146.879
A3	19.948	A18	1507.33
A4	33.345	A19	11.368
A5	9.462	A20	345
A6	11.652	A21	109.817
A7	1.176	A22	114.555
A8	12.581	A23	1113.373
A9	8.698	A24	11.714
A10	24.847	A25	254.474
A11	2713.042	A26	2708.261
A12	918.575	A27	4.617
A13	1572	A28	381.005
A14	7.952	A29	7.201
A15	41.747	A30	6208

4.6 Performance Metric

The following times were noted down for a workload:

- Time for loading Tl
- Time for processing declarative queries Td
- Time for processing procedural queries Tp
- Time for remaining queries Tr

The performance metric was now calculated as follows:
$(Tl * Td * Tp * Tr)^{1/4}$

4.7 Installation

The installation procedure to be followed for BigBench has been specified step-wise in the Readme file at [10].

Chapter 5

Experiments and Results

5.1 Aim of the Experiment

The aim of the experiments was to benchmark the cluster built using commodity hardware systems and study the results for predictive analysis.

5.2 Setup of Cluster

The cluster setup involved one namenode and multiple datanodes. There is a single resource manager on the same system as the namenode and on each system with a datanode there is one NodeManager.
The configuration of the systems were as follows:

- **Namenode** : [Dual CPU] Intel Xeon E5-2620 v2 @ 2.10GHz server with 8 * 16384 MB @1600 MHz Samsung Synchronous DDR3 RAM and LSI MegaRAID SAS 9240-4i disk with 6 Gb/s SATA on each of 4 internal ports.

- **Datanode** : Intel(R) Core(TM)2 CPU E7500 @ 2.93GHz commodity machine with 2048 MB @800MHz Synchronous DDR RAM and Seagate's 500GB 7200 RPM 3.5" Internal Hard Drive with 16MB Cache and 3 Gb/s SATA.

Apart from the namenode, in one experiment 3 datanodes have been used while in another experiment 4 datanodes have been used.

5.3 Data Generation and Workload

The data is generated using the PDGF generator provided by [10] . The data is loaded onto the cluster first in the form of directories and files as required by Hive and then it is converted into the hive tables. The workload for the experiments has been defined by a collection of 9 queries from the 30 queries defined by BigBench in [11] . The distribution of queries among different types of data is as follows:

Query-Type	Queries	Percentage	Data-Type	Queries	Percentage
Declarative	3,6,7,8	44.4%	Structured	3,6,7,8,9	55.5%
Mixed	2,5,9	33.4%	Semi-Structured	1,2	22.2%
Procedural	1,4	22.2%	Unstructured	4,5	22.3%

The description of the 9 queries in both English and also in SQL-MR based syntax can be found in Appendix A 7:

5.4 Procedure

The following steps describe the procedure used by us to run the workload against different data set sizes (1GB, 5GB, etc.) in the two experiments:

1. The Hadoop Multinode Cluster was setup with as 3 datanodes in one experiment and 4 datanodes in the other experiment.
2. The data was generated in the form of one file for each table on the localmachine using PDGF generator provided by BigBench. The default data set size is 1GB. The appropriate scale-factor was provided using the "-sf“ command line option in order to generate larger data sets.
3. The files were copied onto the hadoop file system in the directory format required by Hive tables.
4. Hive was populated with the data from the files on the HDFS.
5. Each HiveQL query was run on the data and the response time reported by the system was noted down.

The average query response time for each data size is obtained by using the following formula:

$$avg\_response\_time = \frac{\sum_{i=1}^{9} repsonsetime_i}{9}$$
$$where\ i = Query\ Number$$

The scale factor for n GB data size can be obtained using the formula given below:

$$scale\ factor\ for\ n\ GB\ data\ size = \frac{query\ response\ time\ for\ n\ GB\ data}{query\ response\ time\ for\ 1\ GB\ data} \tag{5.1}$$

Chapter 6

Results

The following table lists the average query response time for different data sizes obtained experimentally:

Table 6.1: Query Response Time for 3 and 4 Data nodes

Data Size	Query Response Time (sec)	
	3 DataNodes	4 DataNodes
1	188	172
5	317	275
10	617	530
25	1154	1040
40	1451	1682
45	1793	1924
50	1956	2205
75	3054	3029
100	4491	4312

The figure 6.1(a) and 6.1(b) indicate the comparison between the average query response times for 3 datanodes and 4 datanodes.

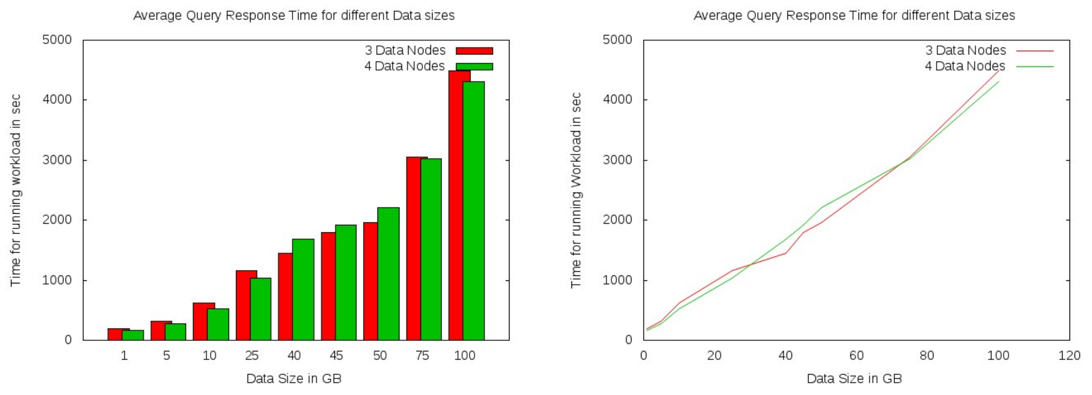


Figure 6.1: Average Query Response Times for different data sizes

On observing the above graphs, we notice that intially, the average query response time using 4 datanodes was less than that using 3 datanodes. However, gradually, as the data size increased, we see opposite behavior i.e. the average response time for 4

datanodes was more in comparison to that for 3 datanodes. Furthermore, when the data size was increased further, the average response time behavior becomes same as it was initally.
The average query response time is affected by a combination of multiple factors. The execution of queries on HDFS system involves division and distribution of work across multiple nodes. The intermediate results are transferred across the system, combined and compiled to produce the final result. Initially, when the data size is small, the intermediate results produced are also small in size. So the time consumed for transfer over the network is not a significant factor. However, with increase in data size, the size of the intermediate results is also found to be large. Also, it is observed that intensive amount of swapping takes place at each node. Thus, the time consumed for transfer over a larger network and that utilized in swapping dominates over faster computation due to distribution of workload across more number of nodes. This results in the change of behavior as observed in the graph. On further increasing the data size, we observe that the effect of intensive swapping becomes constant when the swap space is utilized to its maximum limit. Hence, distribution of workload across more nodes becomes the dominating factor in comparison to the network transfer time because the intermediate results produced are large in size and have similar effect even if the network size is increased. As a result, computations are faster and the behavior of average response time becomes same as it was earlier.

The following table lists the scale factor for change in query response time with respect to that of 1 GB data:

Table 6.2: Scale Factor for change in query response time for 3 and 4 Data nodes

Data Size	Scale Factor (w.r.t 1GB)	
	3 DataNodes	4 DataNodes
1	1	1
5	1.59	1.68
10	3.08	3.28
25	6.04	6.13
40	9.78	7.71
45	11.18	9.53
50	12.82	10.40
75	17.61	16.24
100	25.06	23.88

The figures 6.2(a) and 6.2(b) indicate the above parameter for 3 datanodes and 4 datanodes respectively.
As shown in the graph above, we have drawn a mean line which can be extrapolated to predict the scale factors for larger data sizes. The predicted response times thus obtained for the data sizes used in the experiment and their relative errors have been listed in the tables below.
Table 6.3 is for 3 datanodes while table 6.4 is for 4 datanodes.

Table 6.3: Error in Predicted time w.r.t Experimental time for 3 Datanodes

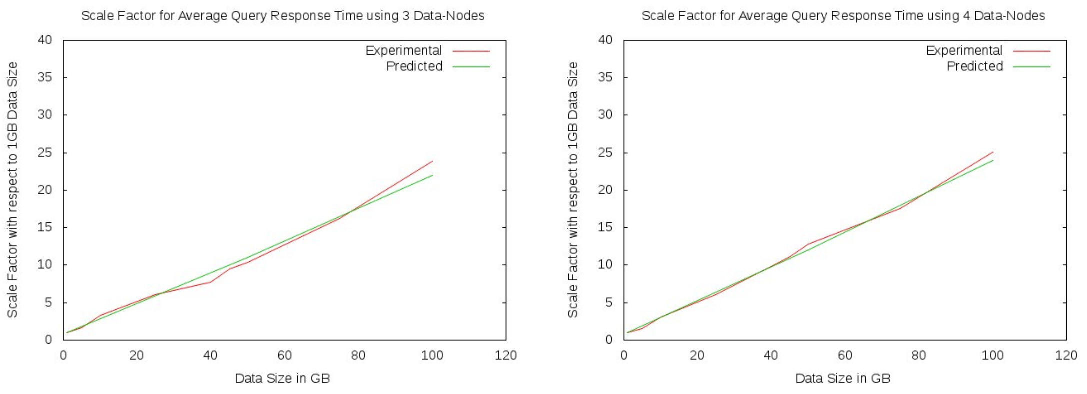


Figure 6.2: Scale Factor for change in query response time w.r.t. 1 GB data

Data Size	Experimental Time (sec)	Predicted Time (sec)	Deviation (sec)
1	188	188	0
5	317	338	21
10	617	526	91
25	1154	1090	64
40	1451	1654	103
45	1793	1842	49
50	1956	2030	74
75	3054	2970	84
100	4491	3910	481
Mean Deviation			107

Table 6.4: Error in Predicted time w.r.t Experimental time for 4 Data nodes

Data Size	Experimental Time (sec)	Predicted Time (sec)	Deviation (sec)
1	172	170	2
5	275	321	46
10	530	510	20
25	1040	1078	38
40	1682	1646	32
45	1924	1835	79
50	2205	2024	181
75	3029	2970	59
100	4312	3916	396
Mean Deviation			94

As noted from the above table, graphical analysis can be used to predict the average query response time with acceptable devaition upto a certain extent.

Chapter 7

Conclusion and Future Work

In this experiment, we have designed a specific workload for benchmarking a HDFS system built using commodity machines. We have also presented the rationale behind the pattern of system response time observed during the experiment as we changed the size of the cluster. Based on the experimental results obtained, we have used graphical analysis to predict the expected response time for larger data sets.

In future, we plan to expand the size of the cluster furthermore and work on developing a graphical analysis to predict the optimum size of cluster required to work with data of a particular size. Also, we intend to make an open source release of the customized benchmarking suite used in the experiment available to the Big Data community.

References

[1] "Hadoop Architecture. Available at http://docs.hortonworks.com. Accessed on 30th June 2014," November 2011.

[2] "Hadoop Mapreduce. Available at http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html. Accessed on 30th June 2014," March 2013.

[3] J. Lin and C. Dyer, *Data Intensive Text-Processing with MapReduce.* Morgan and Claypool Synthesis Lectures, 2010.

[4] "SingleNode Installation. Available at http://hadoop.apache.org/docs/r0.18.2. Accessed on 7 June 2014."

[5] "MultiNode Installation. Available at http://solaimurugan.blogspot.in/2013/11/setup-multi-node-hadoop-20-cluster.html. Accessed on 5 June 2014," March 2013.

[6] "Ganglia Installation. Available at http://www.slashroot.in/how-install-and-configure-ganglia-gmod-and-ganglia-gmetad. Accessed on 26th June 2014," March 2013.

[7] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive - A Warehousing Solution Over a Map-Reduce Framework," *Proceedings of the VLDB Endowment*, pp. 1626–1629, August 2009.

[8] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen, "BigBench: towards an industry standard benchmark for big data analytics," *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 1197–1208, 2013.

[9] D. Laney, "3D Data Management : Controlling Data Volume, Velocity and Variety," *Technical report, Meta Group*, 2001.

[10] "BigBench Installation
https://github.com/intel-hadoop/Big-Bench/blob/master/README.md. Accessed on July 2, 2014."

[11] T. Rabl, M. Poess, C. Baru, and H.-A. Jacobsen, "Specifying big data benchmarks,"

Appendix A

This appendix lists the 9 queries in english and sql-mr.

- **Query 1**. Find the last 5 products that are mostly viewed before a given product was purchased online. Only products in certain categories and viewed within 10 days before the purchase date are considered.

```
SELECT lastviewed_item , purchased_item , COUNT (*)
FROM nPath ( ON web_clickstreams
PARTITION BY wcs_user_sk
ORDER BY wcs_click_date_sk ,wcs_click_time_sk
MODE ( NONOVERLAPPING )
PATTERN (   A +. B   )
SYMBOLS ( true AS A , wcs_sales_sk IS NOT NULL AS B )
RESULT (
LAST ( wcs_item_sk OF A ) AS lastviewed_item ,
LAST ( wcs_click_date_sk OF A ) AS lastviewed_date ,
FIRST ( wcs_item_sk OF B ) AS purchased_item ,
FIRST ( wcs_click_date_sk OF B) AS purchased_date
)
)
WHERE purchased_item = 16891
AND purchased_date - lastviewed_date < 11
GROUP BY 1 ,2;
```

- **Query 2**. For online sales, compare the total sales in which customers checked online reviews before making the purchase and that of sales in which customers did not read reviews. Consider only online sales for a specific category in a given year.

```
BEGIN ;
DROP VIEW clicks ;
CREATE VIEW clicks AS (
SELECT c.wcs_item_sk AS item ,
c.wcs_user_sk AS uid ,
c.wcs_click_date_sk AS c_date ,
c.wcs_click_time_sk AS c_time ,
c.wcs_sales_sk AS sales_sk ,
w.wp_type AS wpt
FROM web_clickstreams c , web_page w
WHERE c.wcs_web_page_sk = w.wp_web_page_sk
and c.wcs_user_sk IS NOT NULL
);
DROP VIEW sales_review;
CREATE VIEW sales_review AS (
SELECT s_sk
FROM nPath ( ON clicks
PARTITION BY uid
ORDER BY c_date , c_time
MODE ( NONOVERLAPPING )
PATTERN (    A  +. C *. B     )
SYMBOLS ( wpt =     review      AS A , TRUE AS C ,
sales_sk IS NOT NULL AS B )
RESULT ( FIRST ( c_date OF B ) AS s_date ,
FIRST ( sales_sk OF B ) AS s_sk ) )
WHERE s_date > 2451424 AND s_date <2451424+365
);
SELECT SUM ( CASE WHEN ws.ws_sk IN
( SELECT * FROM sales_review)
THEN ws_net_paid
ELSE 0 END ) AS review_sales_amount ,
SUM ( ws_net_paid) -
SUM ( CASE WHEN ws.ws_sk IN
( SELECT * FROM sales_review)
THEN ws_net_paid
ELSE 0 END ) AS no_review_sales_amount
FROM web_sales ws
WHERE ws.ws_sold_date_sk > 2451424
AND ws.w s_sold_date_sk <2451424+365;
END ;
```

- **Query 3**. (TPC-DS 48) Calculate the total sales by different types of customers (e.g., based on marital status, education status), sales price and different combinations of state and sales profit.

```
SELECT SUM ( ss_quantity)
FROM store_sales , store , customer_demographics ,
customer_address , date_dim
WHERE s_store_sk =ss_store_sk
AND ss_sold_date_sk = d_date_sk
AND d_year = 1998
AND (( cd_demo_sk = ss_cdemo_sk
AND cd_marital_status =    M
AND cd_education_status =    4  yr Degree
AND ss_sales_price between 100.00 AND 150.00)
OR
( cd_demo_sk = ss_cdemo_sk
AND cd_marital_status =    M
AND cd_education_status =    4  yr Degree
AND ss_sales_price between 50.00 AND 100.00)
OR
( cd_demo_sk = ss_cdemo_sk
AND cd_marital_status =    M
AND cd_education_status =    4  yr Degree
AND ss_sales_price between 150.00 AND 200.00) )
AND ((ss_addr_sk = ca_address_sk
AND ca_country =     United States
AND ca_state in (   K Y   ,   G A   ,   N M   )
AND ss_net_profit between 0 AND 2000)
OR
( ss_addr_sk = ca_address_sk
AND ca_country =   U n i t e d  S t a t e s
AND ca_state in (   M T   ,   O R   ,   I N   )
AND ss_net_profit between 150 AND 3000)
OR
( ss_addr_sk = ca_address_sk
AND ca_country =   U n i t e d  S t a t e s
AND ca_state in (   W I   ,   M O   ,   W V   )
AND ss_net_profit between 50 AND 25000) ) ;
```

- **Query 4**. For all products, extract sentences from its product reviews that contain positive or negative sentiment and display the sentiment polarity of the extracted sentences.

```
SELECT pr_item_sk , out_content ,
out_polarity , out_sentiment_words
FROM ExtractSentiment
( ON product_reviews100
TEXT_COLUMN ( p r _ r e v i e w _ c o n t e n t )
MODEL (  d i c t i o n a r y  )
LEVEL (    sentence    )
ACCUMULATE (  p r _ i t e m _ s k  )
)
WHERE out_polarity =    N E G
OR out_polarity =    P O S   ;
```

- **Query 5**. For a given product, measure the correlation of sentiments, including the number of reviews and average review ratings, on product monthly revenues.

```
BEGIN ;
DROP VIEW IF EXISTS review_stats;
CREATE VIEW review_stats AS (
SELECT p.pr_item_sk AS pid ,
CAST ( p.r_count AS INT ) AS reviews_count ,
CAST ( p.avg_rating AS INT ) AS avg_rating ,
CAST ( s.revenue AS INT ) AS m_revenue
FROM ( SELECT pr_item_sk , COUNT (*) AS r_count ,
AVG ( pr_review_rating) AS avg_rating
FROM product_reviews
WHERE pr_item_sk IS NOT NULL
GROUP BY 1) p
JOIN
( SELECT ws_item_sk , SUM (ws_net_paid) AS revenue
FROM web_sales
WHERE ws_sold_date_sk > 2452642 -30
AND ws_sold_date_sk < 2452642
AND ws_item_sk IS NOT NULL
GROUP BY 1) s
ON p.pr_item_sk=s.ws_item_sk) ;
SELECT *
FROM corr_reduce ( ON
corr_map ( ON
review_stats
COLUMNS (    [ m_revenue:reviews_count] ,[ m_revenue:avg_rating]
KEY_NAME (    k     )
)
PARTITION BY k) ;
DROP VIEW review_stats;
END ;
```

- **Query 6**. (TPC-DS 90) What is the ratio between the number of items sold over the internet in the morning (8 to 9am) to the number of items sold in the evening (7 to 8pm) of customers with a specified number of dependents. Consider only websites with a high amount of content.

```
SELECT CAST (amc AS DECIMAL (15 ,4)) / CAST ( pmc AS DECIMAL (15,4)
FROM ( SELECT COUNT (*) amc
FROM web_sales,household_demographics , time_dim , web_page wp
WHERE ws_sold_time_sk = time_dim.t_time_sk
AND ws_ship_hdemo_sk = household_demographics.hd_demo_sk
AND ws_web_page_sk = wp.wp_web_page_sk
AND time_dim.t_hour BETWEEN 8 AND 8+1
AND household_demographics.hd_dep_count = 5
AND wp . w p _ c h a r _ c o u n t BETWEEN 5000 AND 5200) at ,
( SELECT COUNT (*) pmc
FROM web_sales , h o u s e h o l d _ d e m o g r a p h i c s , time
WHERE w s _ s o l d _ t i m e _ s k = time_dim . t _ t i m e _ s k
AND w s _ s h i p _ h d e m o _ s k = h o u s e h o l d _ d e m o g
AND w s _ w e b _ p a g e _ s k = wp . w p _ w e b _ p a g e _ s k
AND time_dim . t_hour BETWEEN 19 AND 19+1
AND h o u s e h o l d _ d e m o g r a p h i c s. h d _ d e p _ c o
AND wp . w p _ c h a r _ c o u n t BETWEEN 5000 AND 5200) pt
ORDER BY a m _ p m _ r a t i o ;
```

- **Query 7**. (TPC-DS 61) Find the ratio of items sold with and without promotions in a given month and year. Only items in certain categories sold to customers living in a specific time zone are considered.

```
SELECT promotions , total ,
CAST ( p r o m o t i o n s AS DECIMAL (15 ,4) ) /
CAST ( total AS DECIMAL (15 ,4) ) * 100
FROM ( SELECT SUM ( s s _ e x t _ s a l e s _ p r i c e) p r o m
FROM store_sales , store , promotion , date_dim ,
customer , customer_address , item
WHERE s s _ s o l d _ d a t e _ s k = d _ d a t e _ s k
AND s s _ s t o r e _ s k = s _ s t o r e _ s k
AND s s _ p r o m o _ s k = p _ p r o m o _ s k
AND s s _ c u s t o m e r _ s k= c _ c u s t o m e r _ s k
AND c a _ a d d r e s s _ s k = c _ c u r r e n t _ a d d r _ s k
AND s s _ i t e m _ s k = i _ i t e m _ s k
AND c a _ g m t _ o f f s e t = -7
AND i _ c a t e g o r y =     Jewelry
AND ( p _ c h a n n e l _ d m a i l =    Y      OR p _ c h a n n e
OR p _ c h a n n e l _ t v =    Y     )
AND s _ g m t _ o f f s e t = -7
AND d_year = 2001
AND d_moy = 12) p r o m o t i o n a l_ sal es ,
( SELECT sum ( s s _ e x t _ s a l e s _ p r i c e) total
FROM store_sales , store , date_dim ,
customer , customer_address , item
WHERE s s _ s o l d _ d a t e _ s k = d _ d a t e _ s k
AND s s _ s t o r e _ s k = s _ s t o r e _ s k
AND s s _ c u s t o m e r _ s k= c _ c u s t o m e r _ s k
AND c a _ a d d r e s s _ s k = c _ c u r r e n t _ a d d r _ s k
AND s s _ i t e m _ s k = i _ i t e m _ s k
AND c a _ g m t _ o f f s e t = -7
AND i _ c a t e g o r y =     Jewelry
AND s _ g m t _ o f f s e t = -7
AND d_year = 2001
AND d_moy = 12) a l l _ s a l e s
ORDER BY promotions , total ;
```

- **Query 8**. For a given product, measure the effect of competitors prices on products in-store and online sales. (Compute the cross-price elasticity of demand for a given product).

```
BEGIN ;
CREATE VIEW c o m p e t i t o r _ p r i c e _ v i e w AS
( SELECT i_item_sk , ( i m p _ c o m p e t i t o r _ p r i c e -
/ i _ c u r r e n t _ p r i c e AS price_change , imp_start_date
i m p _ e n d _ d a t e - i m p _ s t a r t _ d a t e AS no_days
FROM item , i t e m _ m a r k e t p r i c e s
WHERE i m p _ i t e m _ s k = i _ i t e m _ s k
AND i _ i t e m _ s k in (7 ,17)
AND i m p _ c o m p e t i t o r _ p r i c e < i _ c u r r e n t _
CREATE VIEW s e l f _ w s _ v i e w AS
```

```
( SELECT ws_item_sk ,
SUM ( CASE WHEN w s _ s o l d _ d a t e _ s k >= c . i m p _ s t a
AND w s _ s o l d _ d a t e _ s k < c . i m p _ s t a r t _ d a t e
THEN w s _ q u a n t i t y ELSE 0 END ) AS current_ws ,
SUM ( CASE WHEN w s _ s o l d _ d a t e _ s k >= c . i m p _ s t a
AND w s _ s o l d _ d a t e _ s k < c . i m p _ s t a r t _ d a t e
THEN w s _ q u a n t i t y ELSE 0 END ) AS prev_ws
FROM web_sales , c o m p e t i t o r _ p r i c e _ v i e w c
WHERE w s _ i t e m _ s k = c . i _ i t e m _ s k
GROUP BY 1) ;
CREATE VIEW s e l f _ s s _ v i e w
( SELECT ss_item_sk ,
SUM ( CASE WHEN
AND
THEN
SUM ( CASE WHEN
AND
AS
s s _ s o l d _ d a t e _ s k >= c . i m p _ s t a r t _ d a t e
s s _ s o l d _ d a t e _ s k < c . i m p _ s t a r t _ d a t e + c
s s _ q u a n t i t y ELSE 0 END ) AS current_ss ,
s s _ s o l d _ d a t e _ s k >= c . i m p _ s t a r t _ d a t e -
ss_sold_date_sk < c. imp_start_date190
THEN s s _ q u a n t i t y ELSE 0 END ) AS prev_ss
FROM store_sales , c o m p e t i t o r _ p r i c e _ v i e w c
WHERE c . i _ i t e m _ s k = s s _ i t e m _ s k
GROUP BY 1) ;
SELECT i_item_sk , ( c u r r e n t _ s s + current_ws - prev_ss - p
/ (( prev_ss + prev_ws ) * p r i c e _ c h a n g e) AS c r o s s _
FROM c o m p e t i t o r _ p r ic e_ vi ew , self_ws_view , s e l f
WHERE i _ i t e m _ s k = w s _ i t e m _ s k
AND i _ i t e m _ s k = s s _ i t e m _ s k;
DROP VIEW s e l f _ w s _ v i e w;
DROP VIEW s e l f _ s s _ v i e w;
DROP VIEW c o m p e t i t o r _ p r i c e _ v i e w;
END ;
```

- **Query 9**.Perform category affinity analysis for products purchased online together.

```
CREATE VIEW c_affinity_input AS
( SELECT i.i_category_id AS category_cd ,
s.ws_bill_customer_sk AS customer_id
FROM web_saless INNER JOIN item i
ON s.ws_item_sk=i_item_sk
WHERE i.i_category_id IS NOT NULL);
SELECT *
FROM cfilter ( ON
( SELECT 1)
PARTITION BY 1
DATABASE ( benchmark )
USERID ( benchmark )
PASSWORD ( benchmark )
INPUTTABLE ( benchmark.c_affinity_input )
OUTPUTTABLE ( c_affinity_out )
DROPTABLE ( true )
INPUTCOLUMNS ( category_cd )
JOINCOLUMNS ( customer_id ));
SELECT * FROM c_affinity_out;
DROP TABLE IF EXISTS c_affinity_out;
DROP VIEW IF EXISTS c_affinity_input;
```