BigDataBench User Manual

ICT, Chinese Academy of Sciences

Contacts (Email):
Prof. Jianfeng Zhan,
zhanjianfeng@ict.ac.cn

Table of Contents

1 Introduction

1.1 Context

As a multi-discipline—e.g., system, architecture, and data management—research effort, BigDataBench is a big data benchmark suite. It includes 6 real-world and 2 synthetic data sets, and 32 big data workloads, covering micro and application benchmarks from areas of search engine, social networks, e-commerce, multimedia and bioinformatics. In generating representative and variety of big data workloads, BigDataBench focuses on units of computation frequently appearing in Cloud "OLTP", OLAP, interactive and offline analytics. BigDataBench also provides several (parallel) big data generation tools-BDGS- to generate scalable big data, e.g. PB scale, from small-scale real-world data while preserving their original characteristics. For example, on an 8-node cluster system, BDGS generates 10 TB data in 5 hours. For the same workloads, different implementations are provided. Currently, we and other developers implemented the offline analytics workloads using MapReduce, MPI, Spark, DataMPI, interactive analytics and OLAP workloads using Shark, Impala, and Hive.

1.2 Environment

This document presents user manual information on BigDataBench – including a brief introduction and the setting up guidelines of big data software stacks, and operating guide of all workloads in BigDataBench. The information and specifications contained are for researchers who are interested in big data benchmarking.
Note that the user manual information in the following passage are tested in the environment as follows.
Recommended browner: IE or Chrome.
Recommended OS: Centos 6.0 or later.
Libraries:
JDK 1.6 or later.
C compiler, such as gcc.
C++ compiler, such as g++.
OpenSSH.

1.3 Format specification

The following typographic conventions are used in this user manual:

Convention	Description
Bold	Bold for emphasis.
Italic	Italic for fold and file names.
$command	$command for command lines.
Contents	Contents for contents in configuration files.
`Courier font`	Courier font for screen output.
`Footnote`	Some exception explanations are put in footnote.

2 Overview of Software Packages and Workloads

Software stacks	**Supported workloads**
Hadoop [section 3.1]	MicroBenchmark(Sort, Grep, WordCount) [section 4.2.1] PageRank [section 4.2.2] Index [section 4.2.3] Recommendation [section 4.4.3] NaiveBayes [section 4.4.4]
Spark [section 3.3]	MicroBenchmark(Sort, Grep, WordCount) [section 4.2.1] PageRank [section 4.2.2]] NaiveBayes [section 4.4.4]
MPI [section 3.4]	MicroBenchmark(Sort, Grep, WordCount) [section 4.2.1] PageRank [section 4.2.2] NaiveBayes [section 4.4.4] BFS [section 4.3.1] K-means [section 4.3.2] CC [section 4.3.3] SIFT [section 4.5.2] DBN [section 4.5.3] Speech Recognition [section 4.5.4] Ray Tracing [section 4.5.5] Image Segmentation [section 4.5.6] Face Detection [section 4.5.7] SAND [section 4.6.1] BLAST [section 4.6.2]
Hive [section 3.5]	Select Query, Aggregation Query, Join Query [section 4.4.1] Aggregation, Cross Product, Difference, Filter, OrderBy, Project, U- nion [section 4.4.2]
Impala [section 3.7]	Select Query, Aggregation Query, Join Query [section 4.4.1] Aggregation, Cross Product, Difference, Filter, OrderBy, Project, Union [section 4.4.2]

3 Installation and Configuration of Software

3.1 Setting up Hadoop

Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models

Step 0: Prerequisites
Java JDK: version 1.6 or later
OpenSSH Hadoop: we recommend version 1.2.1, which was used and tested in our environment.

Step 1: Download Hadoop
Download the binary one: hadoop-1.2.1-bin.tar.gz from:
`https://dist.apache.org/repos/dist/release/hadoop/common/hadoop-1.2.1/`

Step 2: Basic Configuration
Step 2.1 Setup passphraseless ssh
Master node must ssh to slave nodes without a passphrase.
If you use a standalone mode, the master and slave nodes are the same one.
If you cannot ssh to nodes without a passphrase, execute the following commands at slave nodes:

```
$ ssh-keygen -t dsa -f $HOME/.ssh/id_dsa -P ""
```

This should result in two files, $HOME/.ssh/id_dsa (private key) and $HOME/.ssh/id_dsa.pub (public key).
Copy $HOME/.ssh/id_dsa.pub to Master nodes. On **slave nodes** run the following commands[1]:

```
$ cat id_dsa.pub » $HOME/.ssh/authorized_keys
$ chmod 0600 $HOME/.ssh/authorized_keys
```

On the **master node** test the results by ssh'ing to **slave nodes:**

[1] Depending on the version of OpenSSH the following commands may also be required:
$ cat id_dsa.pub » $HOME/.ssh/authorized_keys2
$ chmod 0600 $HOME/.ssh/authorized_keys2
An alternative is to create a link from authorized_keys2 to authorized_keys:
$ cd $HOME/.ssh && ln -s authorized_keys2 authorized_keys

```
$ ssh -i $HOME/.ssh/id_dsa server
```

Step 2.2 Configure Hadoop
Decompress the Hadoop package.

```
$ tar -zxvf hadoop-1.2.1.tar.gz
```

Edit the configuration file:

```
$ cd hadoop-1.2.1/conf
```

In hadoop-env.sh:
Add:
```
export JAVA_HOME=/path/to/java_home
```
In core-site.xml:
Add:
```
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://master_node_hostname:9100</value>
</property>
</configuration>
```
In hdfs-site.xml:
Add:
```
<configuration>
<property>
<name>dfs.name.dir</name>
<value>/path/to/store/metadata</value>
<description> </description>
</property>
<property>
<name>dfs.data.dir</name>
<value>/path/to/store/hdfs_data</value>
<description> </description>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
</configuration>
```
In mapred-site.xml
Add:

```
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>master_node_hostname:9200</value>
</property>
</configuration>
```

In master
Add:

```
master hostname
```

In slave
Add:

```
slave hostname
```

Add the Hadoop home path to the environment variable of the system.
Add:

```
$ vim ~/.bashrc
```

```
export HADOOP_HOME=/path/to/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
```

```
$ source ~/.bashrc
```

Then scp all this package (i.e., hadoop-1.2.1) to all slave nodes and put them in the same directory.

Step 3: Start Hadoop
1.Format the HDFS:

```
$ cd hadoop-1.0.2
$ bin/hadoop namenode -format
```

2.Start Hadoop:

```
$ bin/start-all.sh
```

Step 4: Stop Hadoop

```
$ bin/stop-all.sh
```

3.2 Setting up hadoop-2.0.0-cdh4.2.0

Cloudera CDH is the world's most complete, tested, and popular distribution of Apache Hadoop and related projects.

Step 0: Prerequisites

CentOS: 6.5
Java JDK: version 1.6 or later
Open-ssh:

Step 1: Download and Install hadoop-2.0.0-cdh4.2.0

1.Download the hadoop-cdh from cloudera website. We recommend hadoop-2.0.0-cdh4.2.0 (`http://archive.cloudera.com/cdh4/cdh/4/hadoop-2.0.0-cdh4.2.0.tar.gz`), which was used and tested in our environment.
2.Unpack the tarball.

```
$tar -xzvf hadoop-2.0.0-cdh4.2.0.tar.gz
```

3.Set environment variable HADOOP_HOME (/path/to/hadoop-2.0.0-cdh4.2.0), add HADOOP_HOME, YARN_HOME, HADOOP_HDFS_HOME to your PATH.

```
$vim ~/.bash_profile
```

In the ~/.bash_profile: Add:

```
export HADOOP_HOME=/path/to/hadoop-2.0.0-cdh4.2.0
export YARN_HOME=/path/to/hadoop-2.0.0-cdh4.2.0
export HADOOP_HDFS_HOME=/path/to/hadoop-2.0.0-cdh4.2.0
export PATH=$HADOOP_HOME/bin:$PATH
```

Make ~/.bash_profile effective:

```
$source ~/.bash_profile
```

Step 2: Basic Configuration
1.Enter the directory of configuration file.

```
$cd $HADOOP_HOME/etc/hadoop/
```

2.Configure hadoop-env.sh.

```
$vim Hadoop-env.sh
```

In the hadoop-env.sh:
Add: JAVA_HOME
3.Configure core-site.xml. Here, we use pseudo-distributed mode as example.

```
$vim core-site.xml
```

In the *core-site.xml*:
Add:

```
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>/home/hadoop_file/tmp</value>
</property>
```

4.Configure hdfs-site.xml.

```
$vim hdfs-site.xml
```

In the *hdfs-site.xml* : Add:

```
<property>
<name>dfs.name.dir</name>
<value>/home/hadoop_file/name</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>/home/hadoop_file/data</value>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
```

5.Configure *yarn-site.xml.*

```
$vim yarn-site.xml
```

In the *yarn-site.xml*:
Add:

```
<property>
<name>yarn.resoucemanager.address</name>
<value>localhost:8032</value>
</property>
<property>
<name>yarn.resourcemanager.scheduler.address</name>
<value>localhost:8030</value>
</property>
<property>
<name>yarn.resourcemanager.resource-tracker.address</name>
<value>localhost:8031</value>
</property>
<property>
<name>yarn.resourcemanager.admin.address</name>
<value>localhost:8033</value>
</property>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce.shuffle</value>
</property>
```

Step 3: Install Native Lib
1.Install hadoop-2.0.0+922-1.cdh4.2.0.p0.12.el6.x86_64.rpm from website: http://archive.cloudera.com/cdh4/redhat/6/x86_64/cdh/4.2.0/RPMS/x86_64/hadoop-2.0.0+922-1.cdh4.2.0.p0.12.el6.x86_64.rpm
2.Unpack the rpmball.

```
$rpm2cpio hadoop-2.0.0+922-1.cdh4.2.0.p0.12.el6.x86_64.rpm | cpio -div
```

3.In the current unpacked directory, enter the directory /usr/lib/Hadoop/lib/native and generate native.tar.gz.

```
$pwd
$cd 'pwd'/usr/lib/Hadoop/lib/native
$rm libhadoop.so
$rm libsnappy.so
$rm libsnappy.so.1
```

```
$ln -s libhadoop.so.1.0.0 libhadoop.so
$ln -s libsnappy.so.1.1.3 libsnappy.so.1
$ln -s libsnappy.so.1.1.3 libsnappy.so
$cd ..
$tar -zcf native.tar.gz native
```

Step 3: Start hadoop-2.0.0-cdh4.2.0
1.Format the file system.

```
$cd $HADOOP_HOME/bin
$./hadoop namenode -format
```

2.Start hadoop processes after formatted successfully.

```
$cd $HADOOP_HOME/sbin
$./start-all.sh
```

Step 4: Test hadoop-2.0.0-cdh4.2.0

Step 5: Stop hadoop-2.0.0-cdh4.2.0

3.3 Setting up Spark

Step 0: Prerequisites
Java JDK: version 1.6 or later
Scala: version 2.10.4 or later
OpenSSH
Hadoop: version 1.2.1
Spark: The latest version of Spark our benchmark support is version 1.3.0. The benchmark may be executable on later version of Spark, however, we do not test all the newer version. So we recommend version 1.3.0, which was used and tested in our environment.

Step 1: Download Spark
Download the prebuild one: spark-1.3.0-bin-hadoop1.tgz from: `http://www.apache.org/dyn/closer.lua/spark/spark-1.3.0/spark-1.3.0-bin-hadoop1.tgz`

Step 2: Basic Configuration
Step 2.1 Setup passphraseless ssh
Master node must ssh to work nodes without a passphrase.
If you use a standalone mode, the master and work nodes are the same one.
If you cannot ssh to nodes without a passphrase, execute the following commands at **worker nodes**:

```
$ ssh-keygen -t dsa -f $HOME/.ssh/id_dsa -P ""
```

This should result in two files, $HOME/.ssh/id_dsa (private key) and $HOME/.ssh/id_dsa.pub (public key).
Copy $HOME/.ssh/id_dsa.pub to Master nodes. On **work nodes** run the following commands[2]:

```
$ cat id_dsa.pub » $HOME/.ssh/authorized_keys
$ chmod 0600 $HOME/.ssh/authorized_keys
```

On the **master node** test the results by ssh'ing to **worker nodes**:

```
$ ssh -i $HOME/.ssh/id_dsa server
```

Step 2.2 Configure Spark Decompress the Spark package.

```
$ tar -zxvf spark-1.3.0-bin-hadoop1.tgz
```

Edit the configuration file:

```
$ cd spark-1.3.0/conf
$ cp spark-env.sh.template spark-env.sh
```

In spark-env.sh
Add:
SPARK_MASTER_IP= MASTER_HOSTNAME

[2] Depending on the version of OpenSSH the following commands may also be required:
$ cat id_dsa.pub » $HOME/.ssh/authorized_keys2
$ chmod 0600 $HOME/.ssh/authorized_keys2
An alternative is to create a link from authorized_keys2 to authorized_keys:
$ cd $HOME/.ssh && ln -s authorized_keys2 authorized_keys

```
$ cp spark-defaults.conf.template spark-defaults.conf
```

In spark-defaults.conf
Add:
spark.master spark://MASTER_HOSTNAME:7077
In slaves:
Add:
WORK_HOSTNAME #each work per line
Add the Spark home path to the environment variable of the system.

```
$ vim  /.bashrc
```

Add:
export SPARK_HOME=/path/to/spark
export PATH=$PATH:$SPARK_HOME/sbin

```
$ source  /.bashrc
```

Step 3: Start Spark

```
$ cd spark-1.3.0/
$ sbin/start-all.sh
```

Step 4: Stop Spark

```
$sbin/stop-all.sh
```

3.4 Setting up MPI

Setting up Software MPICH2
MPICH2 is a portable implementation of the MPI2.2 standard. In this manual, we use the version of mpich2-1.5, for you own installation, you can also choose a higher version.

Step 0: Prerequisites
Required:
a C compiler, such as gcc.
a C++ compiler, such as g++

Step 1: Download mpich2
Download links for the latest stable release can always be found on
https://www.mpich.org/downloads/
If you want to download the version of mpich2-1.5.tar.gz, you can download at
http://www.mpich.org/static/downloads/1.5/

Step 2: Basic Installation
Step 2.1 Unpack the tar file

```
$ tar -zxvf mpich2-1.5.tar.gz
$ cd mpich2-1.5
```

Step 2.2 Configure
Choosing an non-existent or empty installation directory, such as /home/mpich2-ins; Command "echo $SHELL" to know the current shell your terminal program used, we use CentOS operating system and bash shell;
For shell of bash and sh, using the following command to configure:[3]

```
$./configure –prefix=/home/mpich2-ins 2>$1 | tee c.txt
```

Step 2.3 Build
Build command:

```
$make 2>$1 | tee m.txt
```

Step 2.4 Intall
Install command:

```
$make install 2>$1 | tee mi.txt
```

Step 2.5 Add the bin subdirectory to the PATH environment variable
For shell of bash and sh, using the command:

```
$vim ~/.bashrc
```

export PATH=$PATH:/home/mpich2-ins/bin
save and exit vim

[3] Note that if you don't have a fortran compile and neednâĂŹt to build any Fortran programs, you can disable Fortran support using –disable-f77 and –disable-fc.

```
$source ~/.bashrc
```

Step 3: Check
Step 3.1 Checking the path
Using the command to display the path to your bin subdirectory:

```
$which mpicc
$which mpic++
```

In our example, the first command should display
/home/mpich2-ins/bin/mpicc
Step 3.2 Checking the location on all machines
The installation directory on all machines should be the same. One method is to install mpich2 on one machine and share its installation directory with other machines, the other method is to install mpich2 on all machines with the same installation directory.

Step 4: Use MPICH2 to run programs
Step 4.1 Go into the example directory
In the installation package, such as mpich2-1.5.tar.gz, there is an example directory to test.
Step 4.2 Compile an example C program using mpicc
Using the command to produce corresponding executable file:

```
$mpicc cpi.c -o cpi
```

this command will produce an executable file named cpi
Step 4.3 Run the program using multiple processes on one or more machines
Using the command to run the program on local machine:

```
$mpirun -n 4 ./cpi
```

Note that the number followed -n is the number of processes, here 4 means four processes.
Using the command to run the program on multiple machines;

```
$vim machine_file the machine_file contains the information of all machines
```

one example of machine_file, including 3 nodes named node1, node2 and node3:
node1
node2
node3
save and exit vim

```
$mpirun -f machine_file -n 3 ./cpi
```

Note: -f parameter specificies the machine information, and -n parameter specificies the process number After typing the above mpirun command, the terminal will display the process information and the value of pi, such as

```
Process 0 of 3 is on node1
Process 1 of 3 is on node2
Process 2 of 3 is on node3
pi is approximately 3.1415926544231239, Error is XXX
wall clock time = XXX
```

3.5 Setting up Hive

Hive facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL.

Step 0: Prerequisites
Java JDK: version 1.6 or later
Hadoop: we recommend version 1.2.1, which was used and tested in our environment.

Step 1: Download and Install Hive
1.Download the most recent stable release of Hive from one of the Apache download mirrors. We recommend version 0.9.0 (`http://archive.apache.org/dist/hive/hive-0.9.0/`), which was used and tested in our environment.
2.Unpack the tarball.

```
$tar -xzvf hive-0.9.0.tar.gz
```

3.Set environment variable HIVE_HOME(/path/to/hive-0.9.0), add $HIVE_HOME/bin to your PATH.

```
$vim ~/.bash_profile
```

In the ~/.bash_profile:
Add:

```
export HIVE_HOME=/path/to/hive-0.9.0
export PATH=$HIVE_HOME/bin:$PATH
```

Step 2: Basic Configuration
1.Copy the configuration file from template.

```
$ cd $HIVE_HOME/conf
$ cp hive-env.sh.template hive-env.sh
$ cp hive-default.xml.template hive-site.xml
```

2.Configure $HIVE_HOME/conf/hive-env.sh.
In hive-env.sh:
Add:

```
HADOOP_HOME=$HADOOP_HOME
export HIVE_CONF_DIR=$HIVE_HOME/conf
export HIVE_AUX_JARS_PATH=$HIVE_HOME/lib
```

Make hive-env.sh effective:

```
$source hive-env.sh
```

3.Create the following directory to save the hive relevant data on hdfs:

```
$HADOOP_HOME/bin/hadoop fs -mkdir /tmp
$HADOOP_HOME/bin/hadoop fs -mkdir /user/hive/warehouse
$HADOOP_HOME/bin/hadoop fs -chmod g+w /tmp
$HADOOP_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse
```

Step 3: Start Hive
1.Make sure that you have successfully started Hadoop.
2.Type the following at the command line to start running hive, and enter Hive CLI.

```
$HIVE_HOME/bin/hive
```

Step 4: Test Hive
In Hive CLI, Type the following command to test whether Hive have been successfully installed. If return 'OK', install Hive successfully.

```
hive > show tables;
```

Step 5: Stop Hive In Hive CLI, Type the following command:

```
hive > exit;
```

3.6 Setting up hive-0.10.0-cdh4.2.0(for Impala)

Step 0: Prerequisites **CentOS**: 6.5
Java JDK: version 1.6 or later
Hadoop: hadoop-2.0.0-cdh4.2.0
Mysql: 5 or later

Step 1: Download and Install hive-0.10.0-cdh4.2.0
1.Download the hive-cdh from cloudera website. We recommend hive-0.10.0-cdh4.2.0 (`http://archive.cloudera.com/cdh4/cdh/4/hive-0.10.0-cdh4.2.0.tar.gz`), which was used and tested in our environment.
2.Unpack the tarball.

```
$tar -xzvf hive-0.10.0-cdh4.2.0.tar.gz
```

3.Set environment variableÂăHIVE_HOME (/path/to/hive-0.10.0-cdh4.2.0), addÂăHIVE_HOME to yourÂăPATH.

```
$vim ~/.bash_profile
```

In the ~/.bash_profile:
Add:
export HIVE_HOME=/path/to/hive-0.10.0-cdh4.2.0
export PATH=$HIVE_HOME/bin:$PATH Make ~/.bash_profile effective:

```
$source ~/.bash_profile
```

Step 2: Basic Configuration
1.Enter the directory of configuration file.

```
$cd $HIVE_HOME/conf
$ cp hive-env.sh.template hive-env.sh
$ cp hive-default.xml.template hive-site.xml
```

2.Configure $HIVE_HOME/conf/hive-env.sh.
In hive-env.sh:
Add:
HADOOP_HOME=$HADOOP_HOME
export HIVE_CONF_DIR=$HIVE_HOME/conf
export HIVE_AUX_JARS_PATH=$HIVE_HOME/lib
Make *hive-env.sh* effective:

```
$source hive-env.sh
```

3.Download mysql-connector-java.jar, and transfer mysql-connector-java.jar to $HIVE_HOME/lib.

```
$wget 'http://mirrors.sohu.com/mysql/Connector-J/mysql-connector
-java-5.1.35.tar.gz'
$ tar xzf mysql-connector-java-5.1.35.tar.gz
$ cp mysql-connector-java-5.1.35-bin.jar $HIVE_HOME/lib
```

4.After Starting Mysql, establish appropriate MySQL account for Hive, and give sufficient authority.

```
$mysql -uroot -phadoophive
mysql>CREATE DATABASE metastore;
mysql>USE metastore;
mysql>SOURCE /usr/lib/hive/scripts/metastore/upgrade/mysql/hive
-schema-0.10.0.mysql.sql;
mysql> CREATE USER 'hive'@'%' IDENTIFIED BY 'hadoophive';
mysql>CREATE USER 'hive'@'localhost' IDENTIFIED BY 'hadoophive';
mysql>REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'hive'@'%';
mysql>REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'hive'@'localhost';
mysql>GRANT SELECT,INSERT,UPDATE,DELETE,LOCK TABLES,EXECUTE ON
metastore.* TO 'hive'@'%';
mysql>GRANT SELECT,INSERT,UPDATE,DELETE,LOCK TABLES,EXECUTE ON
metastore.* TO 'hive'@'localhost';
mysql>FLUSH PRIVILEGES;
mysql> quit;
```

5.Configure Hive_HOME/hive-site.xml to Integrate Mysql as the metadata of Hive.

```
$sudo vim $HIVE_HOME/conf/hive-site.xml
```

In the *core-site.xml*:
Modify:

```
<!–?xml version="1.0"?–>
<!–?xml-stylesheet type="text/xsl" href="configuration.xsl"?–>
<configuration>
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://localhost:3306/metastore?createDatabaseIfNotExist=true</value>
<description>the URL of the MySQL database</description>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.jdbc.Driver</value>
</property>
<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>hive</value>
</property>
<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>hadoophive</value>
</property>
</configuration>
```

```
$ sudo service hive-metastore start
$ sudo service hive-server start
$ sudo -u hive hive
```

Step 3: Start Hive-metastore and Hive-server

Step 4: Test Hive
In Hive CLI, Type the following command to test whether Hive have been successfully installed. If return 'OK', install Hive successfully.

```
hive > show tables;
```

Step 5: Stop Hive
In Hive CLI, Type the following command:

```
hive > exit;
```

3.7 Setting up Impala

Step 0: Prerequisites
CentOS: 6.5
Java JDK: version 1.6 or later
Hadoop: hadoop-2.0.0-cdh4.2.0
Hive: hive-0.10.0-cdh4.2.0
MySQL:5 or later
Note: the version of cdh, hive and impala need to match; impala requires some specific linux version. The details can be found in official document, which is shown in http://www.cloudera.com/content/cloudera-content/cloudera-docs/Impala/latest/PDF/Installing-and-Using-Impala.pdf.

Step 1: Download and install Impala
1.Download the all rpm package from website: http://archive.cloudera.com/impala/redhat/6/x86_64/impala/1/RPMS/x86_64. Here, we use impala-1.0.1 in our environment. There rpm packages include:
impala-1.0-1.p0.819.el6.x86_ 64.rpm, impala-debuginfo-1.0-1.p0.819.el6.x86_ 64.rpm, impala-server-1.0-1.p0.819.el6.x86_ 64.rpm, impala-shell-1.0-1.p0.819.el6.x86_ 64.rpm, impala-state-store-1.0-1.p0.819.el6.x86_ 64.rpm
2.Download bigtop-utils-0.4+300-1.cdh4.0.1.p0.1.el6.noarch.rpm from: http://archive.cloudera.com/impala/redhat/6/x86_64/impala/1/RPMS/noarch/.
3.Download libevent-1.4.13-4.el6.x86_64.rpm from http://rpm.pbone.net/index.php3?stat=26&dist=79&size=67428&name=libevent-1.4.13-4.el6.x86_64.rpm
4.Install rpm packages in datanode and hive node.

```
 $rpm -ivh bigtop-utils-0.4+300-1.cdh4.0.1.p0.1.el6.noarch.rpm
$rpm -ivh libevent-1.4.13-4.el6.x86_ 64.rpm
$rpm -ivh impala-1.0-1.p0.819.el6.x86_ 64.rpm
$rpm -ivh impala-server-1.0-1.p0.819.el6.x86_ 64.rpm
$rpm -ivh impala-server-1.0-1.p0.819.el6.x86_ 64.rpm
$rpm -ivh impala-shell-1.0-1.p0.819.el6.x86_ 64.rpm
$rpm -ivh impala-debuginfo-1.0-1.p0.819.el6.x86_ 64.rpm
```

Step 2: Basic Configuration
1.Copy hive-site.xml, core-site.xml and hdfs-site.xml to the default directory of configuration file /etc/impala/conf.

```
 $cp $HIVE_HOME/conf/hive-site.xml /etc/impala/conf/hive-site.sh
$cp $HADOOP_HOME/etc/hadoop/core-site.xml /etc/impala/conf/
core-site.xml
$cp $HADOOP_HOME/etc/hadoop/hdfs-site.xml /etc/impala/conf/
hdfs-site.xml
```

2.In the datanode, configure /etc/impala/conf/hive-site.xml.

```
$cd /etc/impala/conf
```

In hive-site.xml, modify the mysql address.
Modify:

```
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://localhost:3306/hive?createDatabaseIfNotExist=true</value>
<description>JDBC connect string for a JDBC metastore</description>
</property>
```

3.In all impala node, configure /etc/impala/conf/core-site.xml.
In core-site.xml,
Add:

```
<property>
<name>dfs.client.read.shortcircuit</name>
<value>true</value>
</property>
<property>
<name>dfs.client.read.shortcircuit.skip.checksum</name>
<value>false</value>
</property>
```

Modify:

```
<property>
<name>fs.defaultFS</name>
<value>hdfs://172.18.11.206:12900</value>
</property>
```

Note: *172.18.11.206* is the ip address of the test impala node.
4.In all impala node, configure /etc/impala/conf/hdfs-site.xml.
In hdfs-site.xml,
Add:

```
<property>
<name>dfs.client.read.shortcircuit</name>
<value>true</value>
</property>
<property>
```

```
<name>dfs.domain.socket.path</name>
<value>/var/run/hadoop-hdfs/dn._PORT</value>
</property>
<property>
<name>dfs.datanode.hdfs-blocks-metadata.enabled</name>
<value>true</value>
</property>
<property>
<name>dfs.client.file-block-storage-locations.timeout</name>Âă
<value>10000</value>
</property>
```

5.In all impala node, modify /etc/default/impala.

```
IMPALA_STATE_STORE_HOST=172.18.11.206
IMPALA_STATE_STORE_PORT=24000
IMPALA_BACKEND_PORT=22000
IMPALA_LOG_DIR=/var/log/impala
IMPALA_STATE_STORE_ARGS=“-log_dir=${IMPALA_LOG_DIR} state_store_port=${IMPALA
IMPALA_SERVER_ARGS=“\
-log_dir=$IMPALA_LOG_DIR \
-state_store_port=$IMPALA_STATE_STORE_PORT \
-use_statestore \
-state_store_host=$IMPALA_STATE_STORE_HOST\
-be_port=$IMPALA_BACKEND_PORT"
ENABLE_CORE_DUMPS=false
LIBHDFS_OPTS=-Djava.library.path=/usr/lib/impala/lib
MYSQL_CONNECTOR_JAR=$HIVE_HOME/lib/mysql-connector-java-5.1.35.jar
IMPALA_BIN=/usr/lib/impala/sbin
IMPALA_HOME=/usr/lib/impala
HIVE_HOME=$HIVE_HOME
#HBASE_HOME=/usr/lib/hbase
IMPALA_CONF_DIR=/etc/impala/conf
HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
HIVE_CONF_DIR=$HIVE_HOME/conf
#HBASE_CONF_DIR=/etc/impala/conf
```

Step 3: Start Impala

```
$ sudo service impala-state-store restart
$ sudo service impala-server restart
```

Step 4: Test Impala
1.Use the follow command to view the start status of impala.

```
$ps -ef |grep impala
```

2.Enter the impala shell chient.

```
$ impala-shell
```

The Impala shell information will be printed on the screen:

```
Welcome to the Impala shell. Press TAB twice to see a list of available
comma Copyright (c) 2012 Cloudera, Inc. All rights reserved. (Shell
build version: Impala Shell v1.0.1 (df844fb) built on Tue Jun 4 08:0813)
[Not connected] >
```

In Impala CLI, input the follow command to connect the impala node.

```
[Not connected] > connect 172.18.11.206
```

The connect information will be printed on the screen:

```
Connected to 172.18.11.206:21000 Server version: impalad version 1.0.1
RELEASE (build df844fb967cec8740f08d3527ef)
```

Step 5: Stop Impala
In Impala CLI, Type the following command:

```
 [172.18.11.206:21000] > exit;
```

3.8 Setting up Mysql

MySQL is an open source relational database management system (RDBMS).

Step 0: Prerequisites
CentOS: 6.5

```
$sudo yum install mysql-server
```

Step 1: Install Mysql by YUM

```
$sudo /usr/bin/mysql_secure_installation
```

Step 2: Initialize Mysql service The information of initialization will be printed on the screen:

```
[...]
Enter current password for root (enter for none): press âĂŸenterâĂŹ key
OK, successfully used password, moving on...
[...]
Set root password? [Y/n] Y
New password:hadoophive
Re-enter new password:hadoophive
Remove anonymous users? [Y/n] Y
[...]
Disallow root login remotely? [Y/n] N
[...]
Remove test database and access to it [Y/n] Y
[...]
Reload privilege tables now? [Y/n] Y
All done!
```

Step 3: Test Mysql service
1.Enter Mysql CLI.

```
$mysql -uroot -phadoophive
```

2.In Mysql CLI, Type the following command to test whether Hive have been successfully installed. If return 'OK', install Hive successfully.

```
mysql>show table;
```

4 Workloads

4.1 BDGS

The BigDatabenchmark is accompanied by a Big Data generation tools, called BDGS (Big Data Generator Suite). It is a comprehensive suite developed to generate synthetic big data while preserving their 4V properties. It can generate Text, Graph and Table data.

Specifically, our BDGS can generate data using a sequence of three steps. First, BDGS selects application-specific and representative real-world data sets. Second, it constructs data generation models and derives their parameters and configurations from the data sets. Finally, given a big data system to b e tested, BDGS generates synthetic data sets that can b e used as inputs of application specific workloads. In the release edition, BDGS consist of three parts: Text generator, Graph generator, and Table generator. We now intro duce how to use these tools to generate data.

4.1.1 Get BDGS

The BDGS has been packaged in each the benchmark suite, users do not need to download separately. User can find it from the any of the following benchmark packages:
http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.1_Hadoop.tar.gz
http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.1.5_Spark.tar.gz

4.1.2 Compile BDGS

The BDGS is pre-compiled, and if it is not compatible with users' system, users can compile it by the following ways:
Pre-required software
The BDGS depends **gls**, if the systems do not have the package installed. Users can execute the prepar.sh script in each benchmark directory.
Such as in **BigDataBench_V3.1_Hadoop.tar.gz**

```
$ cd BigDataBench_V3.1_Hadoop_Hive/
$ ./prepar.sh
```

Compile Text data generate:
Cd to the directory and execute make command:

```
$ cd BigDataGeneratorSuite/Text_ datagen
$ make
```

Compile Graph data generate:
Cd to the directory and execute make command:

```
$ cd BigDataGeneratorSuite/Graph_ datagen
$ make
```

If there are some error about the incompatible of Snap when executes make command, users need to recompile the **snap-core** and update the Snap.O:

```
$ cd snap-core
$ make
$ mv Snap.o ../
```

And the execute the make command under directory of BigDataGenerator-Suite/Graph_datagen again:

```
$ cd ../ $ make
```

Compile Table data generate:
Cd to the directory and execute make command:

```
$ cd BigDataGeneratorSuite/Table_ datagen/personal_ generator $ make
```

4.1.3 Generate data

How to generate data will be explained in "**Prepare the input**" section of each workload running instruction.

After generating the big data, we integrate a series of workloads to process the data in our big data benchmarks. In this part, we will introduce how to run the Benchmark for each workload. It typically consists of two steps. The first step is to generate the big data and the second step is to run the applications using the generated data.

4.2 Search Engine

In Search Engine domain, we have used data sets including :Wikipedia Entries, Google Web Graph, ProfSearch Resumes and SoGou Data. The Wikipedia Entries are used by WordCount, Sort, Grep, Index workloads. The Google Web Graph is used by PageRank workload. ProfSearch Resumes are used by Cloud OLTP (Write,Read,Scan).The SoGou Data is used by Nutch Server.

4.2.1 MicroBenchmark(Sort, Grep, WordCount)

MicroBenchmark is a program suite, which include sort, grep and wordcount. The instruction of these three workloads is similar, so we put them together.
Hadoop based
1. Required Software Stacks
Hadoop
BGDS
2. Get workloads from BigDataBenchmark
Download the benchmark package BigDataBench_V3.1_Hadoop.tar.gz from http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.1_Hadoop.tar.gz
Decompress the Hadoop package.

```
$ tar -zxvf BigDataBench_V3.1_Hadoop.tar.gz
```

Prepare:

```
$ cd BigDataBench_V3.1_Hadoop_Hive/
$ ./prepar.sh
```

3. Prepare the input

```
$ cd MicroBenchmarks/
$./genData_MicroBenchmarks.sh
```

Then you will be asked how many data you like to generate:

```
./genData_MicroBenchmarks.sh
Preparing MicroBenchmarks data dir WORK_DIR=BigDataBench_V3.1
_Hadoop_Hive/MicroBenchmarks data will be put in BigDataBench
_V3.1_Hadoop_Hive/MicroBenchmarks/data-MicroBenchmarks/in
print data size GB : (enter a number here)
```

4. Run the workload

```
$ ./run_MicroBenchmarks.sh
```

Then you will be asked which workload to run:

```
WORK_DIR= BigDataBench_V3.1_Hadoop_Hive/MicroBenchmarks data should
be put in BigDataBench_V3.1_Hadoop_Hive/MicroBenchmarks/data-MicroBenchmarks/in
Please select a number to choose the corresponding Workload algorithm
1. sort Workload
2. grep Workload
3. wordcount Workload
Enter your choice : (enter the corresponding number)
```

5. Collect the running results
The output of the workload will be put in hdfs with location $pwd /data-MicroBenchmarks/out/wordcount

Spark based
1. Required Software Stacks
Spark
BGDS
2. Get workloads from BigDataBenchmark
Download the benchmark package BigDataBench_V3.1.5_Spark.tar.gz from `http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.1.5_Spark.tar.gz` Decompress the Hadoop package.

```
$ tar -zxvf BigDataBench_V3.1.5_Spark.tar.gz
```

Add the JAR file to environment variable:
In ~/.bashrc Add:
export JAR_FILE = /path/to/BigDataBench_V3.1.5_Spark/JAR_FILE/bigdatabench-spark_1.3.0-hadoop_1.0.4.jar

```
$source ~/.bashrc
```

3. Prepare the input

```
$ cd BigDataBench_V3.1.5_Spark/MicroBenchmarks
$ ./genData_MicroBenchmarks.sh
```

Then you will be asked how many data you like to generate:

```
Preparing MicroBenchmarks data dir
WORK_DIR=/BigDataBench_V3.1.5_Spark/MicroBenchmarks data will be put
```

```
in /BigDataBench_V3.1.5_Spark/MicroBenchmarks/data-MicroBenchmarks
print data size GB : print data size GB : (enter a number here)
```

4. Run the workload

```
$ ./run_ MicroBenchmarks.sh
```

```
 Please select a number to choose the corresponding Workload algorithm
1. Sort Workload
2. Grep Workload
3. Wordcount Workload
Enter your choice : (enter the corresponding number)
```

MPI based

1. Required software stacks
MPICH2
2. Get workload MicroBenchmark from BigDataBench
Download link for MicroBenchmark http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_MPI_V3.1.tar.gz
3. Prepare the input
The data set used by MicroBench is generated by big data generation tool (BDGS). To generate data:
1) Unpack the downloaded tar file

```
$tar -zxvf BigDataBench_ MPI_ V3.1.tar.gz
```

2) Generate data
For Sort:

```
$cd BigDataBench_ MPI_ V3.1/MicroBenchmark/MPI_ Sort
$sh genData_ sort.sh
```

For Grep:

```
$cd BigDataBench_ MPI_ V3.1/MicroBenchmark/MPI_ Grep
$sh genData_ grep.sh
```

For WordCount:

```
$cd BigDataBench_ MPI_ V3.1/MicroBenchmark/MPI_ WordCount
$sh genData_ wordcount.sh
```

Input the data size you want to generate with the units of GB, such as 10 if you want to generat 10 GB data. After this step, it will generate *data-sort(/grep/wordcount)/in* under respective directory, such as MPI_Sort(/Grep/Wordcount).

4. Run the workload
Unpack the downloaded tar file

```
$tar -zxvf BigDataBench_MPI_V3.1.tar.gz
```

Install MicroBenchmark

```
$cd BigDataBench_MPI_V3.1/MicroBenchmark/MPI_Sort(/Grep/WordCount)
$make
```

After this step, there will be one executable files named *mpi_sort(/grep/wordcount)* under the current directory. Run the workload
For Sort, command is:

```
$mpirun -f machine_file -n PROCESS_NUM ./mpi_sort input_file output_file
```

For Grep, command is:

```
$mpirun -f machine_file -n PROCESS_NUM ./mpi_grep input_file pattern
```

For WordCount, command is:

```
$mpirun -f machine_file -n PROCESS_NUM ./mpi_wordcount input_file
```

In our example, the three command would be:

```
$mpirun -f machine_file -n 12 ./mpi_sort data-sort/in output
$mpirun -f machine_file -n 12 ./mpi_grep data-grep/in abc
$mpirun -f machine_file -n 12 ./mpi_wordcount data-wordcount/in
```

5. Collect the running results
When the workload run is complete, it will display the running information, such as:
`Total running time:5.000000 sec`
Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6. Note:
For grep workload, the second parameter (pattern) in running command line means the expression needs to be matched.

4.2.2 PageRank

Hadoop based

1. Required Software Stacks

Hadoop
BGDS

2. Get workloads from BigDataBenchmark

Download the benchmark package BigDataBench_V3.1_Hadoop.tar.gz from http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.1_Hadoop.tar.gz

Decompress the Hadoop package.

```
$ tar -zxvf BigDataBench_V3.1_Hadoop.tar.gz
```

Prepare:

```
$ cd BigDataBench_V3.1_Hadoop_Hive/
$ ./prepar.sh
```

3. Prepare the input

```
$ cd SearchEngine/PageRank/
$./genData_PageRank.sh
```

Then you will be asked how many data you like to generate:
`Generate PageRank data`
`Please Enter The Iterations of GenGragh:` *`(enter a number here, It means the number of vertices generated, represented by power of 2)`*

4. Run the workload

```
$ ./run_PageRank.sh #_Iterations_of_GenGragh
```

5 Collect the running results

The output of the workload will be put in hdfs with location: /user/root/pr_vector

Spark based

1. Required Software Stacks

Spark
BGDS

2. Get workloads from BigDataBenchmark

Download the benchmark package BigDataBench_V3.1.5_Spark.tar.gz from http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.1.5_Spark.tar.gz

Decompress the Hadoop package.

```
$ tar -zxvf BigDataBench_V3.1.5_Spark.tar.gz
```

Add the JAR file to environment variable:
In ~/.bashrc Add:
export JAR_FILE = /path/to/BigDataBench_V3.1.5_Spark/JAR_FILE/bigdatabench-spark_1.3.0-hadoop_1.0.4.jar

```
$source ~/.bashrc
```

3. Prepare the input

```
$ cd BigDataBench_V3.1.5_Spark/SearchEngine/Pagerank
$ ./genData_PageRank.sh
```

Then you will be asked how many data you like to generate:
`Generate PageRank data`
`Please Enter The Iterations of GenGragh:` *(enter a number here, It means the number of vertices generated, represented by power of 2)*

4. Run the workload

```
$ ./run_Pagerank.sh
```

Then you will be asked which workload to run:
`Internation I:` *(enter the number input in 3)*
5. Collect the running results The output of the workload will be put in hdfs with location: /spark-pagerank-result

MPI based
MPI_Pagerank is a parallel implementation of pagerank algorithm.
1. Required software stacks
MPICH2
Cmake: Cmake 2.8.12.2 is prefered
Boost1.43.0
When you install the boost packet, make sure that the mpi packet has been installed.

```
$sh bootstrap.sh
$./bjam
```

Building parallel-bgl-0.7.0:

```
$cd BigDataBench_ V3.0_ MPI/SearchEngine/MPI_ Pagerank/parallel-bgl
-0.7.0
$cmake ./
$cd parallel-bgl-0.7.0/libs/graph_ parallel/test
$make distributed_ page_ rank_ test
```

2. Get workload MPI_Pagerank from BigDataBench
Download link for MPI_Pagerank http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_MPI_V3.1.tar.gz
3. Prepare the input
The data set used by MPI_Pagerank is generated by big data generation tool (BDGS).
To generate data:
1) Unpack the downloaded tar file

```
$tar -zxvf BigDataBench_ MPI_ V3.1.tar.gz
$cd BigDataBench_ MPI_ V3.1/SearchEngine/MPI_ Pagerank
```

2) Generate data

```
$sh genData_ PageRank.sh
```

Input the Iterations of GenGragh, after this step, it will generate **data-PageRank** under directory of **BigDataBench_MPI_V3.1/SearchEngine/MPI_Pagerank**.

4. Run the workload
1) Unpack the downloaded tar file

```
$tar -zxvf BigDataBench_ MPI_ V3.1.tar.gz
$cd BigDataBench_ MPI_ V3.1/SearchEngine/MPI_ Pagerank
```

2) Install mpiBLAST
We provide a Compiled executable program named **run_PageRank** under the MPI_Pagerank directory.
3) Run the workload
Run MPI_Pagerank, command is:

```
$mpirun -f machine_ file -n PROCESS_ NUM ./run_ PageRank.sh InputGraph-
file num_ ofVertex num_ ofEdges iterations
```

5. Collect the running results
When the workload run is complete, it will display the running information, such as:

```
INFO: Starting PageRank.
INFO: Params:
InputGraphfile=data-PageRank/Google_genGraph_10.txt,
num_ofVertex=1024, num_ofEdges=2147, iterations=5
256 = 0.813656
...
```

Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6. Note:
The two parameters (num_ofVertex, num_ofEdges) in running command line can be found in standard output when you generate data, such as:

```
[root MPI_Pagerank]# ./genData_PageRank.sh
Generate PageRank data
Please Enter The Iterations of GenGragh: 5
WORK_DIR=/BigDataBench_MPI_V3.1/SearchEngine/MPI_Pagerank data will
be generated in /mnt/sdh/gwl/test/BigDataBench_MPI_V3.1/SearchEngine
/MPI_Pagerank/data-PageRank
sh: gnuplot: command not found
Kronecker graphs. build: 10:42:53, Apr 21 2014. Time: 00:54:50 [Mar
20 2014]
=============================================================================
Output graph file name (-o:)= /mnt/sdh/gwl/test/BigDataBench_MPI_V3.1/
SearchEngine/MPI_Pagerank/data-PageRank/Google_genGraph_5.txt
Matrix (in Maltab notation) (-m:)=0.8305 0.5573; 0.4638 0.3021
Iterations of Kronecker product (-i:)=5
Random seed (0 - time seed) (-s:)=0
*** Seed matrix:
0.8305 0.5573
0.4638 0.3021
(sum:2.1537)
*** Kronecker:
FastKronecker: 32 nodes, 46 edges, Directed...
run time: 0.00s (00:54:50)
```

4.2.3 Index

Hadoop based

1. Required Software Stacks
Hadoop
BGDS
2. Get workloads from BigDataBenchmark
Download the benchmark package BigDataBench_V3.1_Hadoop.tar.gz from http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.

1_Hadoop.tar.gz
Decompress the Hadoop package.

```
$ tar -zxvf BigDataBench_V3.1_Hadoop.tar.gz
```

Prepare:

```
$ cd BigDataBench_V3.1_Hadoop_Hive/
$ ./prepar.sh
```

3. Prepare the input When prepare the input, file *linux.words* and words should exist in dirctory */usr/share/dict.*

```
$ cd /SearchEngine/Index
$./genData_Index.sh
```

Then you will be asked how many data you like to generate:

```
Preparing MicroBenchmarks data dir
WORK_DIR=/BigDataBench_V3.1_Hadoop_Hive/SearchEngine/Index data will
be put in /root/jz/BigDataBench_V3.1_Hadoop_Hive/SearchEngine/Index/data-Index
print data size GB : (enter a number here)
```

4. Run the workload

```
$ ./run_PageRank.sh #_Iterations_of_GenGragh
```

5. Collect the running results
The output of the workload will be put in local directory: result and the ourput is redirected to file: *Index.out*

4.3 Social Networks

In Social Network domain, we have used data set including: Facebook Social Network. The Facebook Social Network is used by CC and Kmeans workloads. The BFS workload data is generated by itself.

4.3.1 Workload MPI_BFS

MPI_BFS is an MPI-based implementation of breadth-first search.
1. Required software stacks
MPICH2
2 Get workload MPI_BFS from BigDataBench
Download link for BFS-MPI.tar.gz http://prof.ict.ac.cn/bdb_uploads/bdb_

3_1/packages/SocialNetwork/BFS-MPI.tar.gz
3. Prepare the input
The data set used by BFS is generated by the program itself.
4. Run the workload
1) Unpack the downloaded tar file

```
$tar -zxvf BFS-MPI.tar.gz
$cd BFS-MPI/graph500/
```

2) Build the MPI executables

```
$vim make.inc
```

set the BUILD_MPI = Yes
change the last line MPICC = XXX -IXXX -LXXX, according to your own MPI installation direcory.
In our example, it should be change to MPICC = /home/mpich2-ins/bin/mpicc -I/home/mpich2-ins/include -L/home/mpich2-ins/lib
Save and exit vim
Using the command to build:

```
$make
```

After this step, there will be two executables files named *graph500_mpi_simple* and *graph500_mpi_one_sided* under directory BFS-MPI/graph500/mpi.
3. Run the workload

```
$cd BFS-MPI/graph500/mpi
$mpirun -f machine_file -n PROCESS_NUM ./graph500_mpi_simple SCALE
edgefactor
```

Note: as previously mentioned (step 4.3), the machine_file contains the node information;
PROCESS_NUM specifies the number of processes;
SCALE and edgefactor are two parameters required by graph500_mpi_simple;
SCALE should be an integer value and specifies the number of vertices to be 2^{SCALE}. This parameter must be provided;
edgefactor is a double value with a default value of 16. It specifies the number of edges to be $(edgefactor * 2^{SCALE})$. This parameter can be omitted.
For example:

```
$mpirun -f machine_file -n 12 ./graph500_mpi_simple 20 15
$mpirun -f machine_file -n 12 ./graph500_mpi_simple 20
```

5. Collect the running results
When the workload run is complete, it will display the running information, such as:

```
SCALE: 20
edgefactor: 15
NBFS: 64
graph_generation: 6.62665 s
num_mpi_processes: 4
construction_time: 54.5597 s
min_time: 0.287835 s
firstquartile_time: 0.288899 s
median_time: 0.292623 s
thirdquartile_time: 0.294162 s
max_time: 0.298326 s
mean_time: 0.29204 s
stddev_time: 0.00283727
min_nedge: 15728430
firstquartile_nedge: 15728430
median_nedge: 15728430
thirdquartile_nedge: 15728430
max_nedge: 15728430
mean_nedge: 15728430
stddev_nedge: 0
min_TEPS: 5.27223e+07 TEPS
firstquartile_TEPS: 5.34685e+07 TEPS
median_TEPS: 5.37498e+07 TEPS
thirdquartile_TEPS: 5.44426e+07 TEPS
max_TEPS: 5.46439e+07 TEPS
harmonic_mean_TEPS: 5.38571e+07 TEPS
harmonic_stddev_TEPS: 65921.9
min_validate: 0.657126 s
firstquartile_validate: 0.66606 s
median_validate: 0.668064 s
thirdquartile_validate: 0.668876 s
max_validate: 0.678658 s
mean_validate: 0.666849 s
stddev_validate: 0.00421153
Steps=: 1470480
```

Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6. Note:

For more information about BFS workload, please refer to: http://www.graph500.org/specifications

4.3.2 Workload MPI_Kmeans

MPI_Kmeans is a mpi-based implementation of kmeans algorithm.
1. Required software stacks
MPICH2
2. Get workload MPI_Kmeans from BigDataBench
Download link for MPI_Kmeans http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_MPI_V3.1.tar.gz
3. Prepare the input
The data set used by MPI_Kmeans is generated by a generating script. To generate data:
1) Unpack the downloaded tar file

```
$tar -zxvf BigDataBench_MPI_V3.1.tar.gz
$cd BigDataBench_MPI_V3.1/SNS/Simple_Kmeans
```

2) Generate data

```
$sh genData_Kmeans.sh
```

Input the data size you want to generate with the units of GB, such as 10 if you want to generat 10 GB data. After this step, it will generate **data-Kmeans** file under directory of **BigDataBench_MPI_V3.1/SNS/Simple_Kmeans**.
4 Run the workload Unpack the downloaded tar file

```
$tar -zxvf BigDataBench_MPI_V3.1.tar.gz
$cd BigDataBench_MPI_V3.1/SNS/Simple_Kmeans
```

Install MPI_Kmeans

```
$make
```

After this step, there will be an executable files named **mpi_main** under the current directory.
Run the workload
Run MPI_Kmeans, command is:

```
$mpirun -f machine_file -n PROCESS_NUM ./mpi_main -i input_file -n cluster_number -o
```

Note: the input_file specifies the name of the input file, such as data-Kmeans;
The cluster_number specifies the number of clusters, such as 5;
-o parameter means output timing results
The coordinates of all cluster centers are writed to file "**data-Kmeans.cluster_centres**", and the membership of all data objects are writed to file "**data-Kmeans.membership**".
5. Collect the running results
When the workload run is complete, it will display the running information, such as:

```
mpi_kmeans is 3.461451 Seconds
Writing coordinates of K=5 cluster centers to file "data-Kmeans.cluster_centres"
Writing membership of N=23000000 data objects to file "data-Kmeans.membership"
Performing **** Simple Kmeans (MPI) ****
Num of processes = 12
Input file: data-Kmeans
numObjs = 23000000
numCoords = 9
numClusters = 5
threshold = 0.0010
I/O time = 100.2676 sec
Computation timing = 3.9639 sec
FPCount=3359518,IntCount=3622512465
```

Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6. Note:
For more information about parameters of mpi_main:
Usage: ./mpi_main [switches] -i filename -n num_clusters
-i filename : file containing data to be clustered
-b : input file is in binary format (default no)
-r : output file in binary format (default no)
-n num_clusters: number of clusters (K must > 1)
-t threshold : threshold value (default 0.0010)
-o : output timing results (default no)
-d : enable debug mode

4.3.3 Workload MPI_ConnectedComponent

MPI_ConnectedComponent is a mpi-based implementation of connected component algorithm.
1. Required software stacks
MPICH2 Cmake: Cmake 2.8.12.2 is prefered Boost1.43.0

When you install the boost packet, make sure that the mpi packet has been installed.

```
$sh bootstrap.sh
$./bjam
```

Building parallel-bgl-0.7.0:

```
$cd BigDataBench_V3.0_MPI/SearchEngine/MPI_Pagerank/parallel-
bgl-0.7.0
$cmake ./
$cd parallel-bgl-0.7.0/libs/graph_parallel/test
$make distributed_page_rank_test
```

2. Get workload MPI_ConnectedComponent from BigDataBench
Download link for MPI_ConnectedComponent http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_MPI_V3.1.tar.gz
3 Prepare the input
The data set used by MPI_ConnectedComponent is generated by data generation tool (BDGS).
To generate data:
1) Unpack the downloaded tar file

```
$tar -zxvf BigDataBench_MPI_V3.1.tar.gz
$cd BigDataBench_MPI_V3.1/SNS/MPI_Connect
```

2) Generate data

```
$sh genData_connectedComponents.sh
```

Input the Iterations of GenGragh, after this step, it will generate **data-Connected_Components** under directory of BigDataBench_MPI_V3.1/SNS /MPI_Connect.
4. Run the workload
Unpack the downloaded tar file

```
$tar -zxvf BigDataBench_MPI_V3.1.tar.gz
$cd BigDataBench_MPI_V3.1/SNS/MPI_Connect
```

Install MPI_ConnectedComponent
We provide a Compiled executable program named run_connectedComponents under the MPI_Connect directory.
Run the workload
Run MPI_ConnectedComponent, command is:

```
$mpirun -f machine_file -n PROCESS_NUM ./run_ connectedComponents InputGraphfile num_ ofVertex num_ ofEdges
```

5. Collect the running results
When the workload run is complete, it will display the running information, such as:

```
INFO: Starting connected components.
INFO: Params: InputGraphfile=amazon_gen_15.txt, num_ofVertex=32768,
num_ofEdges=131655
INFO: Test Complete. components found = 32768, time = 0.97s.
```

Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6. Note:
The two parameters (num_ofVertex, num_ofEdges) in running command line can be found in standard output when you generate data, such as:

```
[root MPI_Pagerank]# ./genData_PageRank.sh
Generate PageRank data
Please Enter The Iterations of GenGragh: 5
WORK_DIR=/BigDataBench_MPI_V3.1/SearchEngine/MPI_Pagerank data will
be generated in /mnt/sdh/gwl/test/BigDataBench_MPI_V3.1/SearchEngine
/MPI_Pagerank/data-PageRank
sh: gnuplot: command not found
Kronecker graphs. build: 10:42:53, Apr 21 2014. Time: 00:54:50 [Mar
20 2014]
==========================================================================
Output graph file name (-o:)= /mnt/sdh/gwl/test/BigDataBench_MPI_V3.1/
SearchEngine/MPI_Pagerank/data-PageRank/Google_genGraph_5.txt
Matrix (in Maltab notation) (-m:)=0.8305 0.5573; 0.4638 0.3021
Iterations of Kronecker product (-i:)=5
Random seed (0 - time seed) (-s:)=0
*** Seed matrix:
0.8305 0.5573
0.4638 0.3021
(sum:2.1537)
*** Kronecker:
FastKronecker: 32 nodes, 46 edges, Directed...
run time: 0.00s (00:54:50)
```

4.4 E-commerce

In E-commerce domain, we have used data set including: E-commerce Transaction Data and Amazon Movie Review. The Amazon Movie Review is used by CF and Bayes workloads. The E-commerce Transaction Data is used by Aggregation Query, Cross Product, Difference, Filter, OrderBy, Project, Union, Select Query, Aggregation Query and Join Query workloads.

4.4.1 Workload - Select Query, Aggregation Query, Join Query

Hive version 1. Required Software Stacks
Java JDK: version 1.6 or later
Hadoop: we recommend version 1.2.1, which was used and tested in our environment.
Hive: we recommend version 0.9.0, which was used and tested in our environment.
2. Get workloads from BigDataBenchmark
Download the Benchmark (BigDataBench_V3.1_Hadoop.tar.gz) form this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.1_Hadoop.tar.gz
3. Prepare the input
Make sure Hadoop and Hive have been successfully started.
Unpack the downloaded tar file:

```
$tar -xzvf BigDataBench_ V3.1_ Hadoop.tar.gz
$cd BigDataBench_ V3.1_ Hadoop_ Hive
```

Set $BigDataBench_Hive_HOME as /path/to/BigDataBench_V3.1_Hadoop_Hive.

```
$ cd Interactive_ Query
$ ./gen_ data.sh
```

4. Run the workload
Run the workloads;

```
$./run_ AnalysiticWorkload.sh
```

The information of selecting workload will be printed on the screen:

```
Please select a number to choose the corresponding Workload algorithm
1. aggregation Workload
2. join Workload
3. select Workload
Enter your choice :
```

For example, we enter **1** to select **aggregation Workload**.
5. Collect the running results
When the workload run is complete, it will display the running information, such as:

```
ok. You chose 1 and we'll use aggregation Workload
WARNING: org.apache.hadoop.metrics.jvm.EventCounter is deprecated. Please
use org.apache.hadoop.log.metrics.EventCounter in all the log4j.properties
files.
Logging initialized using configuration in jar:file:/usr/local/hadoop/
hive-0.9.0/lib/hive-common-0.9.0.jar!/hive-log4j.properties Hive history
file=/tmp/root/hive_job_log_root_201510032145_767144040.txt
OK
Time taken: 4.183 seconds
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size:
1
In order to change the average load for a reducer (in bytes):
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapred.reduce.tasks=<number>
Starting Job = job_201509190338_0009, Tracking URL = http://localhost:
50030/jobdetails.jsp?jobid=job_201509190338_0009
Kill Command = /usr/local/hadoop/hadoop-1.2.1/libexec/../bin/hadoop
job -Dmapred.job.tracker=localhost:9001 -kill job_201509190338_0009
Hadoop job information for Stage-1: number of mappers: 0; number of
reducers: 1
2015-10-03 21:45:59,452 Stage-1 map = 02015-10-03 21:46:06,489 Stage-1
map = 02015-10-03 21:46:09,512 Stage-1 map = 100MapReduce Total cumulative
CPU time: 4 seconds 80 msec
Ended Job = job_201509190338_0009
Moving data to: hdfs://localhost:9000/user/hive/warehouse/tmp27
Table default.tmp27 stats: [num_partitions: 0, num_files: 1, num_rows:
0, total_size: 0, raw_data_size: 0]
MapReduce Jobs Launched:
Job 0: Reduce: 1 Cumulative CPU: 4.08 sec HDFS Read: 0 HDFS Write: 0
SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 80 msec
OK
Time taken: 21.116 seconds
```

6. Notes

Impala version

1. Required Software Stacks
CentOS: 6.5
Java JDK: version 1.6 or later
Hadoop: hadoop-2.0.0-cdh4.2.0
Hive: hive-0.10.0-cdh4.2.0
MySQL: 5.1.73
2. Get workloads from BigDataBenchmark
Download the Benchmark (**BigDataBench_Impala_V3.1.tar.gz**) form this link: `http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_Impala_V3.1.tar.gz`
3. Prepare the input
Make sure Hadoop, Hive and Impala have been successfully started.
Unpack the downloaded tar file:

```
$tar -xzvf BigDataBench_Impala_V3.1.tar.gz
$cd BigDataBench_Impala_V3.1
```

Set $BigDataBench_Impala_HOME as /path/to/ BigDataBench_Impala_V3.1.
In the directory $BigDataBench_Impala_HOME:

```
$ cd Interactive_Query
```

In gen_data.sh:
Add:
BigdataBench_Home=$BigDataBench_Impala_HOME
Execute gen_data.sh:

```
$ ./gen_data.sh
```

The information of selecting data size will be printed on the screen:
`print data size GB :`
For example, we enter 1 to select 1GB data.
4. Run the workload
Modify impala_restart.sh, replace âĂIJyour impala nodeâĂİ with actual impala node ip. For example,
for i in localhost
Run the workloads;

```
$ ./run_AnalysiticWorkload.sh
```

The information of selecting workload will be printed on the screen:
```
Logging initialized using configuration in file:/home/renrui/cdh4/hive-0.10.0-cdh4.2.0/c
```

```
Hive history file=/tmp/root/hive_job_log_root_201510040942_192414277.txt
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/renrui/cdh4/hadoop-2.0.0-cdh4.2.0/share/hadoop/c
SLF4J: Found binding in [jar:file:/home/renrui/cdh4/hive-0.10.0-cdh4.2.0/lib/slf4j-log4j
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
explanation.
OK
Time taken: 3.727 seconds
OK
Time taken: 0.363 seconds
...
```

5. Collect the running results
When the workload run is complete, it will display the running information, such as:
6. Notes

4.4.2 Aggregation, Cross Product, Difference, Filter, OrderBy, Project, Union

Hive version

1. Required Software Stacks
Java JDK: version 1.6 or later
Hadoop: we recommend version 1.2.1, which was used and tested in our environment.
Hive: we recommend version 0.9.0, which was used and tested in our environment.
2. Get workloads from BigDataBenchmark
Download the Benchmark **(BigDataBench_V3.1_Hadoop.tar.gz)** form this link: http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.1_Hadoop.tar.gz
3. Prepare the input
Make sure Hadoop and Hive have been successfully started.
Unpack the downloaded tar file:

```
$tar -xzvf BigDataBench_V3.1_Hadoop.tar.gz
$cd BigDataBench_V3.1_Hadoop_Hive
```

Set $BigDataBench_Hive_HOME as /path/to/BigDataBench_V3.1_Hadoop_Hive.
In the directory $BigDataBench_Hive_HOME/Interactive_MicroBenchmark:

```
$ cd Interactive_MicroBenchmark
$ ./gen_data.sh
```

4. Run the workload
Run the workloads:

```
$ ./run_MicroBenchmarks.sh
```

The information of selecting workload will be printed on the screen:

```
Please select a number to choose the corresponding Workload algorithm
1. aggregationAVG Workload
2. aggregationMAX Workload
3. aggregationMIN Workload
4. aggregationSUM Workload
5. crossproject Workload
6. difference Workload
7. filter Workload
8. orderby Workload
9. projection Workload
10. union Workload
Enter your choice:
```

For example, we enter 5 to select crossproject Workload.

5. Collect the running results

When the workload run is complete, it will display the running information, such as:

```
ok. You chose 5 and we'll use crossproject Workload
WARNING: org.apache.hadoop.metrics.jvm.EventCounter is deprecated. Please
use org.apache.hadoop.log.metrics.EventCounter in all the log4j.properties
files.
Logging initialized using configuration in jar:file:/usr/local/hadoop/hive
-0.9.0/lib/hive-common-0.9.0.jar!/hive-log4j.properties
Hive history file=/tmp/root/hive_job_log_root_201509190403_2134444140.txt
OK
Time taken: 4.117 seconds
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size:
1
In order to change the average load for a reducer (in bytes):
set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapred.reduce.tasks=<number>
Starting Job = job_201509190338_0006, Tracking URL = http://localhost:50030
/jobdetails.jsp?jobid=job_201509190338_0006 Kill Command = /usr/local
/hadoop/hadoop-1.2.1/libexec/../bin/hadoop job -Dmapred.job.tracker=
localhost:9001 -kill job_201509190338_0006
Hadoop job information for Stage-1: number of mappers: 0; number of
reducers: 1
```

```
2015-09-19 04:04:12,519 Stage-1 map = 02015-09-19 04:04:20,569 Stage-1
map = 02015-09-19 04:04:23,600 Stage-1 map = 100MapReduce Total cumulative
CPU time: 4 seconds 130 msec
Ended Job = job_201509190338_0006 Moving data to: hdfs://localhost:9000/user/hive/wareho
Table default.tmp33 stats: [num_partitions: 0, num_files: 1, num_rows:
0, total_size: 0, raw_data_size: 0]
MapReduce Jobs Launched:
Job 0: Reduce: 1 Cumulative CPU: 4.13 sec HDFS Read: 0 HDFS Write: 0
SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 130 msec
OK
Time taken: 22.31 seconds
```

Impala version
1. Required Software Stacks
CentOS: 6.5
Java JDK: version 1.6 or later
Hadoop: hadoop-2.0.0-cdh4.2.0
Hive: hive-0.10.0-cdh4.2.0
MySQL: 5.1.73
2. Get workloads from BigDataBenchmark
Download the Benchmark (BigDataBench_Impala_V3.1.tar.gz) form this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_Impala_V3.1.tar.gz
3. Prepare the input
Make sure Hadoop, Hive and Impala have been successfully started.
Unpack the downloaded tar file:

```
$tar -xzvf BigDataBench_ Impala_ V3.1.tar.gz
$cd BigDataBench_ Impala_ V3.1
```

Set $BigDataBench_Impala_HOME as /path/to/ BigDataBench_Impala_V3.1.
In the directory $BigDataBench_Impala_HOME:

```
$ cd Interactive_ MicroBenchmark
```

In gen_data.sh:
Add:
BigdataBench_Home=$BigDataBench_Impala_HOME
Execute gen_data.sh:

```
$ ./gen_ data.sh
```

4. Run the workload
Modify **impala_restart.sh**, replace âĂIJyour impala nodeâĂİ with actual impala node ip. For example,
for i in localhost
Run the workloads;

```
$ ./run_ MicroBenchmarks.sh
```

The information of selecting workload will be printed on the screen:
Logging initialized using configuration in

```
file:/home/renrui/cdh4/hive-0.10.0-cdh4.2.0/conf/hive-log4j.properties
Hive history file=/tmp/root/hive_job_log_root_201510031047_550088197.txt
SLF4J: Class path contains multiple SLF4J bindings. SLF4J: Found binding
in [jar:file:/home/renrui/cdh4/hadoop-2.0.0-cdh4.2.0/share/hadoop/common
/lib/slf4j-log4j12-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/renrui/cdh4/hive-0.10.0-cdh4.2.0
/lib/slf4j-log4j12-1.6.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
explanation.
OK
Time taken: 3.672 seconds
OK
Time taken: 0.365 seconds
...
```

5. Collect the running results
When the workload run is complete, it will display the running information, such as: 6. Notes

4.4.3 Recommendation Workload

The Recommendation is implemented by the collaborative filtering algorithm. It recommend the certain product to certain costumers.

Hadoop based

1 Required Software Stacks
Hadoop
BDGS
2 Get workloads from BigDataBenchmark
Download the benchmark package BigDataBench_V3.1_Hadoop.tar.gz from `http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.1_Hadoop.tar.gz`
Decompress the Hadoop package.

```
$ tar -zxvf BigDataBench_ V3.1_ Hadoop.tar.gz
```

3 Prepare the input
In E-commerce directory:

```
$ cd E-commerce
$ ./genData_recommendator.sh
```

Then you will be asked how many data you would like to generate:
`Generate E-com-recommendator data`
`Please Enter The Iterations of GenGragh:` *(Enter a number. It means the number of vertices generated, represented by power of 2)* 4 Run the workload
Decompress mahout-distribution-0.6.tar.gz

```
$ tar -zxvf mahout-distribution-0.6.tar.gz
```

Run the workloads

```
$ ./run_recommendator.sh
```

Then you will be asked to enter the number of vertices, which is represented by power of 2:
`Preparing E-com-recommendator data dir`
`WORK_DIR=/.../BigDataBench_V3.1_Hadoop_Hive/E-commerce data should be put in /.../BigDataBench_V3.1_Hadoop_Hive/E-commerce/data-recommendator`
`MAHOUT_HOME=/.../`
`Please Enter The Iterations of GenGragh:` *(Enter the number that is inputted in the prepare stage)*
5 Collect the running results
The output of the workload will be put in **hdfs** with location: ${pwd}/data-recommendator/Amazon_out/out

4.4.4 Workload NaiveBayes

The Naive Bayes is a simple probabilistic classifier, which applies the Bayes' theorem with strong (naive) independency assumptions.
Hadoop based

1 Required Software Stacks
Hadoop
BDGS
2 Get workloads from BigDataBenchmark
Download the benchmark package BigDataBench_V3.1_Hadoop.tar.gz from http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.

1_Hadoop.tar.gz
Decompress the Hadoop package.

```
$ tar -zxvf BigDataBench_V3.1_Hadoop.tar.gz
```

3 Prepare the input
In E-commerce directory:

```
$ cd E-commerce
$ ./genData_naivebayes.sh
```

Then you will be asked how many data you would like to generate:

```
Preparing naivebayes-naivebayes data dir
WORK_DIR=BigDataBench_V3.1_Hadoop_Hive/E-commerce data will be generated
in BigDataBench_V3.1_Hadoop_Hive/E-commerce/data-naivebayes
Preparing naivebayes-naivebayes data dir
print data size GB : (enter a number here)
```

4 Run the workload
Decompress mahout-distribution-0.6.tar.gz

```
$ tar -zxvf mahout-distribution-0.6.tar.gz
```

Run the workloads

```
$ ./run_naivebayes.sh
```

5 Collect the running results
The output will be printed on the screen.

Spark based

1 Required Software Stacks
Spark
BGDS
2 Get workloads from BigDataBenchmark
Download the benchmark package BigDataBench_V3.1.5_Spark.tar.gz from http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_V3.1.5_Spark.tar.gz
Decompress the package.

```
$ tar -zxvf BigDataBench_V3.1.5_Spark.tar.gz
```

Add the JAR file to environment variable:
In ~/.bashrc
Add:
export JAR_FILE = /path/to/BigDataBench_V3.1.5_Spark/JAR_FILE/bigdatabench-spark_1.3.0-hadoop_1.0.4.jar

```
$source ~/.bashrc
```

3 Prepare the input

```
$ cd BigDataBench_V3.1.5_Spark/E-commerce
$ ./ genData_naivebayes.sh
```

Then you will be asked how many data you would like to generate:

```
Preparing naivebayes-naivebayes data dir
WORK_DIR=/root/jz/BigDataBench_V3.1.5_Spark/E-commerce data will be
generated in /root/jz/BigDataBench_V3.1.5_Spark/E-commerce/data-naivebayes
Preparing naivebayes-naivebayes data dir
print data size GB : (enter a number here)
```

4 Run the workload

```
$ ./run_naivebayes.sh
```

5 Collect the running results
The output of the workload will be put in **hdfs** with location: /Bayes-result

MPI based

MPI_NaiveBayes is a mpi-based implementation of naive bayes algorithm.
1. Required software stacks
MPICH2
2. Get workload MPI_NaiveBayes from BigDataBench
Download link for MPI_NaiveBayes `http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BigDataBench_MPI_V3.1.tar.gz`
3. Prepare the input
The data set used by MPI_NaiveBayes is generated by a generating script. To generate data:
1) Unpack the downloaded tar file

```
$tar -zxvf BigDataBench_MPI_V3.1.tar.gz
$cd BigDataBench_MPI_V3.1/E-commerce/MPI_naivebayes
```

2) Generate data

```
$sh genData_naivebayes.sh
```

Input the data size you want to generate with the units of GB, such as 10 if you want to generat 10 GB data. After this step, it will generate **data-naivebayes** under directory of **BigDataBench_MPI_V3.1/E-commerce/MPI_naive bayes**.
4. Run the workload
Unpack the downloaded tar file

```
$tar -zxvf BigDataBench_MPI_V3.1.tar.gz
$cd BigDataBench_MPI_V3.1/E-commerce/MPI_naivebayes
```

Install MPI_NaiveBayes
We provide two executable files **(MPI_NB_train, MPI_NB_predict)** under directory MPI_naivebayes.
Run the workload
To train bayes model, the command is:

```
$mpirun -f machine_file -n PROCESS_NUM ./MPI_NB_train -i input_file
-o train_model
```

To run naive bayes, the command is:

```
$mpirun -f machine_file -n PROCESS_NUM ./MPI_NB_predict
-m train_model -i input_file -o output_file
```

5. Collect the running results
Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.

4.5 Multimedia analytics

In multimedia scenario, we provide seven workloads. Except for BasicMPEG, other six workloads are parallel using MPI. If you want to use several nodes to run the workloads, you must make sure that you have set up **authenticated no-passphrase SSH connections** between these nodes, and make sure that all data sets are put on **all nodes with the same directory (or you can mount a shared directory).**

4.5.1 Workload BasicMPEG

BasicMPEG is a workload undoing the encoding to retrieve original video data.
1 Required software stacks
Libc: This workload is a serial version for present.
2 Get workload BasicMPEG from BigDataBench
Download link for BasicMPEG.tar.gz `http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/BasicMPEG.tar.gz`
3 Prepare the input
The data set used by BasicMPEG is DVD input streams, which can be obtained from `ftp://ftp.tek.com/tv/test/streams/Element/index.html/`
From the above link, download the directory **Element/MPEG-Video** with the subdirectory of "**525**" and "**625**". In the following description of how to run workload BasicMPEG, we assume that you have downloaded the required data set and have put the downloaded directory of "**525**" and "**625**" under the directory /**data/MPEGdec_input**.
4 Run the workload
Unpack the downloaded tar file

```
$tar -zxvf BasicMPEG.tar.gz
```

Build the mpeg2dec and mpeg2enc executables

```
$cd BasicMPEG/MPGdec
$make
$cd ../MPGenc
$make
```

Then you will see **mpeg2dec** and **mpeg2enc** executables under **BasicMPEG/execs** directory. Run the workload

```
$cd BasicMPEG/execs
$vim getPath
```

```
#!/bin/bash
function show_usage {
echo "Usage: $0 source_dir path_file"
exit 1
}
if [ $# -ne 2 ]; then
show_usage
else
if [ -d $1 ]; then
source_dir=$1
else
echo "Invalid source directory"
show_usage
fi
fi
path_file=$2
find $source_dir -type d -name '.*' > ${path_file}.path
```

save and exit vim

```
$sh getPath /data/MPEGdec_input mpeg
```

after this step, you will get a path file "**mpeg.path**" under the directory BasicMPEG/execs

```
$sh batch mpeg.path output
```

Note: the first parameter is the directory data path file (mpeg.path in our example), the second parameter is the directory of output file (output in our example). After this step, you will get an output directory named output.

5 Collect the running results
If you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.

4.5.2 Workload SIFT

SIFT workload is an adaptation of David LoweâĂŹs source code, which detects and describes local features in input images. We modified it to a data parallel version using MPI.
1 Required software stacks
MPICH2
OpenCV package http://sourceforge.net/projects/opencvlibrary/
GDK/GTK+2 http://www.gtk.org/
2 Get workload SIFT from BigDataBench
Download link for Multimedia-MPI.tar.gz http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/Multimedia-MPI.tar.gz
3 Prepare the input
The data set used by SIFT is unstructured images from ImageNet. To get 1 GB

image data: http://prof.ict.ac.cn/bdb_uploads/Media-data/ImageNet_1G.tar.gz
To get 10 GB image data: http://prof.ict.ac.cn/bdb_uploads/Media-data/ImageNet_10G.tar.gz
To get more image data, please visit ImageNet: http://www.image-net.org
Here, we assume that you have downloaded the required image data (such as ImageNet_1G.tar.gz), and have be put under the directory of textbf/data/ImageNet_1G.
4 Run the workload
Unpack the downloaded tar file

```
$tar -zxvf Multimedia-MPI.tar.gz $cd Multimedia-MPI/MicroBenchmark/SIFT
```

Build the MPI executables
Using the command to build:

```
$make
```

After this step, there will be an executable file named **siftfeat_mpi** under directory **SIFT/bin.**
Run the workload
Using getPath script under directory Multimedia-MPI to generate the path of image files:

```
$ sh ../../../getPath /data/ImageNet_1G imagenet_1G
```

Note that the current directory is under SIFT/bin, then the getPath file is under ../../../getPath
After this step, there will be a path file named **imagenet_1G.path** under your **current directory** (SIFT/bin in our example).

```
$mpirun -f machine_file -n PROCESS_NUM ./siftfeat_mpi PATH_FILE
```

Note: as previously mentioned, the machine_file contains the node information;
PROCESS_NUM specifies the number of processes;
PATH_FILE specifies the path of image data generated by genPath.
Type $./siftfeat_mpi -h for more help.
In our example, the command will be:

```
$mpirun -f machine_file -n 12 ./siftfeat_mpi imagenet_1G.path
```

5 Collect the running results
When the workload run is complete, it will display the running information, such as:

```
Loading file and sift begin:
Processing 7851 images, Complete!
```

Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6 Note:
Install GDK/GTK+2: $yum install gtk+*
Install cmake: version 2.8.12.2 or higher
Install OpenCV: $cmake . $make $make install
If when you type $make to make SIFT workload of siftfeat_mpi and get error information "package opencv was not found in the pkg-config search path" after you have installed opencv package, you should add PKG_CONFIG_PATH with the directory of opencv.pc to ~/.bashrc file. For example, we assume that the file opencv.pc is under /usr/local/lib/pkgconfig directory, then you should add the following two sentences in file ~/.bashrc:

```
$vim ~/.bashrc
```

PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
Export PKG_CONFIG_PATH
Save and exit vim

```
$source ~/.bashrc
```

If you type $make to generate siftfeat_mpi file, and get the error information "doxygen: Command not found", you can ignore this error and it will still generate siftfeat_mpi under SIFT/bin.
If you have failed when type $make to generate siftfeat_mpi file, you need to type $make clean before your next make command.

4.5.3 Workload DBN

DBN workload is a MPI implementation of deep belief networks.
1 Required software stacks
MPICH2
2 Get workload DBN from BigDataBench
Download link for Multimedia-MPI.tar.gz `http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/Multimedia-MPI.tar.gz`
3 Prepare the input
The data set used by DBN is MNIST (`http://yann.lecun.com/exdb/mnist/`).
The data set is also packed in Multimedia-MPI.tar.gz, under directory **Multimedia-MPI/DBN/data**.

4 Run the workload
Unpack the downloaded tar file

```
$tar -zxvf Multimedia-MPI.tar.gz
$cd Multimedia-MPI/DBN
```

Build the MPI executables

```
$cd src
```

Using the command to build DBN:

```
$mpic++ DBN.cpp deep.o -o DBN
```

Using the command to build RBM:

```
$mpic++ RBM.cpp deep.o -o RBM
```

Using the command to build StackedRBMS:

```
$mpic++ StackedRBMS.cpp deep.o -o StackedRBMS
```

Using the command to build BP:

```
$mpic++ BP.cpp deep.o -o BP
```

After this step, you will get four executables files named DBN, RBM, StackedRBMS and BP under directory DBN/src, respectively.
Run the workload

```
$cd Multimedia-MPI/DBN/src
```

Run DBN:

```
$mpirun -f machine_file -n PROCESS_NUM ./DBN
```

Run RBM:

```
$mpirun -f machine_file -n PROCESS_NUM ./RBM
```

Run StackedRBMS:

```
$mpirun -f machine_file -n PROCESS_NUM ./StackedRBMS
```

Run BP[4]:

```
$mpirun -f machine_file -n PROCESS_NUM ./BP
```

Note: as previously mentioned, the machine_file contains the node information; PROCESS_NUM specifies the number of processes.

5 Collect the running results
Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6 Note:
For more information about DBN workload, please refer to:
Multmedia-MPI/DBN/README

4.5.4 Workload Speech Recognition

This workload uses CMU sphinx toolkit for speech recognition, we write a parallel version using MPI.
1 Required software stacks
MPICH2
bison-3.0 http://ftp.gnu.org/gnu/bison/
Install: $./configure $make $make install
sphinxbase-0.8, pocketsphinx-0.8, sphinxtrain-1.0.8(optional), cmuclmtk-0.7(optional)
http://sourceforge.net/projects/cmusphinx/files/ Install:
1) sphinxbase-0.8 installation is similar with bison-3.0, add the library path of sphinxbase-0.8 and pkg config path of sphinxbase.pc to ~/.bashrc

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
export LD_LIBRARY_PATH
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH
```

```
$source ~/.bashrc
```

[4] Note that if you want to run BP, you must run StackedRBMS first with the same PROCESS_NUM.

2) pocketsphinx-0.8 installation is similar with bison-3.0
3) cmuclmtk-0.7 installation is similar with bison-3.0
4) sphinxtrain-1.0.8 installation has no make install process.
2 Get workload Speech Recognition from BigDataBench
Download link for Multimedia-MPI.tar.gz http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/Multimedia-MPI.tar.gz
3 Prepare the input
The data set used by Speech Recognition is audio files, to get 1GB audio file:
http://prof.ict.ac.cn/bdb_uploads/Media-data/Audio_1G.tar.gz
To get 10GB audio file:
http://prof.ict.ac.cn/bdb_uploads/Media-data/Audio_10G.tar.gz Here, we assume that you have downloaded the required audio data (such as Audio_1G.tar.gz), and have be put under the directory of /data/Audio_1G.
4 Run the workload
Unpack the downloaded tar file

```
$tar -zxvf Multimedia-MPI.tar.gz
$cd Multimedia-MPI/App/SpeechRecognition
```

Build the MPI executables
Using the command to build:

```
$mpic++ -o decode-mpi-cpp decode-mpi.cpp -DMODELDIR=\"‘pkg-config
–variable=modeldir pocketsphinx‘\" ‘pkg-config –cflags –libs pocketsphinx
sphinxbase’
```

After this step, there will be an executable file named **decode-mpi-cpp** under directory Multimedia-MPI/App/*SpeechRecognition*. Run the workload
Using getPath script under directory Multimedia-MPI to generate the path of audio files:

```
$ sh ../../getPath /data/Audio_1G audio_1G
```

After this step, there will be a path file named **audio_1G.path** under your **current directory** (SpeechRecognition/ in our example).

```
$mpirun -f machine_file -n PROCESS_NUM ./decode-mpi-cpp PATH_FILE
OUTPUT
```

Note: as previously mentioned, the machine_file contains the node information; PROCESS_NUM specifies the number of processes;

PATH_FILE specifies the path file of audio data generated by the script getPath; OUTPUT specifies the output file.
In our example, the command would be:

```
$mpirun -f machine_file -n 12 ./decode-mpi-cpp audio_1G.path output
```

5 Collect the running results
When the workload run is complete, it will display the running information, such as:

```
data outfile end
process=0 time=122.454261
```

Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6 Note:
cmuclmtk-0.7 contains linguistic model, and sphinxtrain-1.0.8 contains acoustic model. If you don't need to use new models, then you don't have to install these two tools.

4.5.5 Workload Ray Tracing

Ray Tracing workload is derived from john.stoneâĂŹs source code, which is a parallel rendering program.
1 Required software stacks
MPICH2
cmake: version 2.8.12.2 or higher
OpenCV
Python: version 2.6.6 or higher
2 Get workload Ray Tracing from BigDataBench
Download link for Multimedia-MPI.tar.gz http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/Multimedia-MPI.tar.gz 3 Prepare the input
The data set used by Ray Tracing is image scene file, to get 1GB image scene file: http://prof.ict.ac.cn/bdb_uploads/Media-data/ImageScene_1G.tar.gz
To get 10G image scene file: http://prof.ict.ac.cn/bdb_uploads/Media-data/ImageScene_10G.tar.gz
To get 100G image scene file: http://prof.ict.ac.cn/bdb_uploads/Media-data/ImageScene_100G.tar.gz
Here, we assume that you have downloaded the required image scene data (such as ImageScene_1G.tar.gz), and have be put under the directory of **/data/ImageScene_1G**.
4 Run the workload
Unpack the downloaded tar file

```
$tar -zxvf Multimedia-MPI.tar.gz
$cd Multimedia-MPI/App/RayTracing
```

Build the MPI executables

```
$cd unix
$make
```

This step will display the architectures, you need to choose one of them, such as **linux-mpi** for parallel version.

```
$make linux-mpi
```

After this step, there will be an executable file named **tachyon** under directory *RayTracing/compile/linux-mpi.* Using getPath script under directory Multimedia-MPI to generate the path of image scene files:

```
$cd Multimedia-MPI/App/RayTracing
$ sh ../../getPath /data/ImageScene_1G imageScene_1G
```

Note that the current directory is under RayTracing, then the getPath file is under ../../getPath
After this step, there will be a path file named imageScene_1G.path under your current directory (RayTracing/ in our example).

```
$sh batch PATH_FILE PROCESS_NUM machine_file
```

Note: as previously mentioned, the machine_file contains the node information;
PROCESS_NUM specifies the number of processes;
PATH_FILE specifies the path of image scene file generated by getPath script.
Please **make sure that the path of executable file tachyon is correct**, if not, you should change it according to your path.
In our example, the command would be:

```
$sh batch imageScene_1G.path 12 machine_file
```

5 Collect the running results
When the workload run is complete, it will display the running information, such as:

```
Scene Parsing Time: 0.0190 seconds
Scene contains 456 objects.
Preprocessing Time: 0.0009 seconds
Rendering Progress: 100Ray Tracing Time: 0.0372 seconds
```

Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6 Note: For more information about Ray Tracing workload, please refer to: `http://jedi.ks.uiuc.edu/~johns/raytracer/`

4.5.6 Workload Image Segmentation

Image Segmentation is an adaptation of Pedro Felipe Felzenszwalb's source code, which segments the input images. We modify it to a data parallel version using MPI.
1 Required software stacks
MPICH2
netpbm: The input of this workload should be PPM format, if you need to convert your own JPEG images to PPM format, you need to install netpbm tool. http://netpbm.sourceforge.net
2 Get workload Image Segmentation from BigDataBench
Download link for Multimedia-MPI.tar.gz http://prof.ict.ac.cn/bdb_uploads/bdb_3_1/packages/Multimedia-MPI.tar.gz
3 Prepare the input
The data set used by Image Segmentation is ImageNet with the PPM image format. So we need to convert JPEG format of ImageNet to PPM format. First, you need to download ImageNet data sets.
To get 1 GB image data:
http://prof.ict.ac.cn/bdb_uploads/Media-data/ImageNet_1G.tar.gz
To get 10 GB image data:
http://prof.ict.ac.cn/bdb_uploads/Media-data/ImageNet_10G.tar.gz
To get more image data, please visit ImageNet:
http://www.image-net.org
Using jpegtopnm command to convert JPEG to PPM format.
Here, we assume that you have downloaded the required image data (such as ImageNet_1G.tar.gz), and have converted them to PPM format (PPM_1G), and put the PPM image data set under the directory of /data/PPM_1G.
4 Run the workload
Unpack the downloaded tar file

```
$tar -zxvf Multimedia-MPI.tar.gz
$cd Multimedia-MPI/App/ImageSegmentation
```

Build the MPI executables
Using the command to build:

```
$make
```

After this step, there will be an executable files named **segment_mpi** under directory *ImageSegmentation/*.
Run the workload
Using getPath script under directory Multimedia-MPI to generate the path of PPM files:

```
$cd Multimedia-MPI/App/ImageSegmentation
$sh ../../getPath /data/PPM_1G ppm_1G
```

Note that the current directory is under ImageSegmentation, then the getPath file is under ../../getPath
After this step, there will be a path file named **ppm_1G.path** under your **current directory** (ImageSegmentation/ in our example).

```
$mpirun -f machine_file -n PROCESS_NUM ./segment_mpi PATH_FILE -o
OUTPUT
```

Note: as previously mentioned, the machine_file contains the node information;
PROCESS_NUM specifies the number of processes;
PATH_FILE specifies the path of PPM image files generated by getPath script;
-o OUTPUT is optional, if this parameter is given, then the segmented files will be stored under OUTPUT directory, if not, there will be no output.
For example:

```
$mpirun -f machine_file -n 12 ./segment_mpi ppm_1G.path
$mpirun -f machine_file -n 12 ./segment_mpi ppm_1G.path out
```

5 Collect the running results
When the workload run is complete, it will display the running information, such as:

```
loading and processing begin
Processing 1942 (5 cannot load) images, Complete!
```

Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6 Note:
An example of shell script for converting JPEG to PPM:

```
$vim JPEGtoPPM.sh
```

```
#!/bin/bash
function show_usage
echo "Usage: $0 source_dir dest_dir"
exit 1

if [ $# -ne 2 ]; then
show_usage
else
```

```
if [ -d $1 ]; then
filedir=$1
else
echo "Invalid source directory"
show_usage
fi
if [ -d $2 ]; then
outdir=$2
else
mkdir $2
outdir=$2
fi
fi
pattern='.JPEG'
for fileOrSubdir in $filedir/*; do
flag=$(echo $fileOrSubdir | grep "$pattern")
temp=$fileOrSubdir/$filedir/$outdir
if [[ "$flag" != "" ]]; then
file=$fileOrSubdir
jpegtopnm $file > $temp/.JPEG/.ppm
else
subdir=$fileOrSubdir
mkdir $temp
for file2 in $subdir/*; do
temp2=$file2/$filedir/$outdir
jpegtopnm $file2 > $temp2/.JPEG/.ppm
done
fi
done
```

Save and exit vim
Using this script:

```
$./JPEGtoPPM.sh <directory of JPEG file> <directory of PPM file>
```

4.5.7 Workload Face Detection

Face Detection workload is an adaptation of flandmark source code, which detects a face in input images. We modify it to a data parallel version using MPI.
1 Required software stacks
MPICH2
cmake: version 2.8.12.2 or higher
OpenCV

2 Get workload Face Detection from BigDataBench
Download link for Multimedia-MPI.tar.gz http://prof.ict.ac.cn/bdb_uploads/

bdb_3_1/packages/Multimedia-MPI.tar.gz

3 Prepare the input
The data set used by BFS is ImageNet. To get 1 GB image data: http://prof.ict.ac.cn/bdb_uploads/Media-data/ImageNet_1G.tar.gz
To get 10 GB image data: http://prof.ict.ac.cn/bdb_uploads/Media-data/ImageNet_10G.tar.gz
To get more image data, please visit ImageNet: http://www.image-net.org
Here, we assume that you have downloaded the required image data (such as ImageNet_1G.tar.gz), and put the image data under the directory of **/data/ImageNet_1G.**

4 Run the workload
Unpack the downloaded tar file

```
$tar -zxvf Multimedia-MPI.tar.gz
$cd Multimedia-MPI/App/FaceDetection
```

Build the MPI executables

```
$cmake .
$make
```

After this step, there will be an executable file named **flandmark_mpi** under directory *FaceDetection/cpp*.
Run the workload
Using getPath script under directory Multimedia-MPI to generate the path of image files:

```
$cd Multimedia-MPI/App/FaceDetection/cpp
$sh ../../../getPath /data/ImageNet_1G ImageNet_1G
```

Note that the current directory is under FaceDetection/cpp, then the getPath file is under ../../../getPath
After this step, there will be a path file named **ImageNet_1G.path** under your **current directory** (FaceDetection/cpp in our example).

```
$mpirun -f machine_file -n PROCESS_NUM ./flandmark_mpi PATH_FILE
-o OUTPUT
```

Note: as previously mentioned, the machine_file contains the node information;
PROCESS_NUM specifies the number of processes;
PATH_FILE specifies the path of image files generated by getPath script;

-o OUTPUT is optional, if this parameter is given, then the detected files will be stored under OUTPUT directory, if not, there will be no output.
For example:

```
$mpirun -f machine_file -n 12 ./flandmark_mpi ImageNet_1G.path $mpirun -f machine_file -n 12 ./flandmark_mpi ImageNet_1G.path out
```

5 Collect the running results
When the workload run is complete, it will display the running information, such as:

```
Structure model loaded in 4 ms.
Faces detected: 1; Detection of facial landmark on all faces took 8 ms
```

Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6 Note:
Face detection must be run under directory FaceDetection/cpp, otherwise, it would report error "Couldnt load Face detector 'haarcascade_frontalface_alt.xml'".

4.6 Bioinformatics

In bioinformatics scenario, we provide two workloads.

4.6.1 Workload SAND

SAND is a workload for genome assembly, which is a part of cooperative computing tools (cctools).
1 Required software stacks
Work Queue platform http://ccl.cse.nd.edu/software/workqueue/
2 Get workload SAND from BigDataBench
Download link for SAND http://ccl.cse.nd.edu/software/files/cctools-5.2.2-source.tar.gz
3 Prepare the input
The data set used by SAND is genome sequence data, which can be obtained from http://ccl.cse.nd.edu/software/sand/
From the above link, download the sample data with four kinds of data size.
4 Run the workload
Unpack the downloaded tar file

```
$tar -zxvf cctools-5.2.2-source.tar.gz
$cd cctools-5.2.2-source
```

Install

```
$./configure –prefix=/SAND_ Workload/cctools
$make
$make install
$export PATH=/SAND_ Workload/cctools/bin:$PATH
```

Then you will see /SAND_Workload/cctools installation directory.
3. Run the workload
1) To generate repeats from a FASTA file

```
$meryl -B -m 24 -C -L 100 -v -o small.meryl -s small.fa
$meryl -Dt -s small.meryl -n 100 > small.repeats
```

2) compress the sequence data into a compressed FASTA (.cfa) file

```
$sand_ compress_ reads small.fa small.cfa
```

3) Start the filtering step

```
$sand_filter_ master -r small.repeats small.cfa small.cand
```

4) Start the alignment step

```
$sand_ align_ master sand_ align_ kernel -e "-q 0.04 -m 40" small.cand small.cfa
small.ovl
```

Note that The options -q 0.04 -m 40 passed to sand_align_kernel indicate a minimum alignment quality of 0.04 and a minimum alignment length of 40 bases. After the sequence alignment step completes, you will have an overlap (.ovl) file that can be fed into the final stages of your assembler to complete the consensus step.
5 Collect the running results
A progress table will be printed to standard out:

```
Total | Workers | Tasks Avg | K-Cand K-Seqs | Total Time | Idle Busy
| Submit Idle Run Done Time | Loaded Loaded | Speedup
0 | 0 0 | 0 0 0 0 0.00 | 0 0 | 0.00 8 | 0 48 | 100 52 48 0 0.00 | 1000
284 | 0.00 10 | 0 86 | 100 13 86 1 7.07 | 1000 284 | 0.71
36 | 1 83 | 181 14 83 2 19.47 | 1810 413 | 1.08
179 | 1 83 | 259 92 83 3 22.51 | 2590 1499 | 0.38
186 | 2 80 | 259 15 80 85 28.54 | 2590 1499 | 13.04
199 | 2 80 | 334 90 80 86 29.96 | 3340 1499 | 12.95
```

```
200 | 2 80 | 334 90 80 114 59.43 | 3340 1499 | 33.88
202 | 2 81 | 334 9 81 165 86.08 | 3340 1499 | 70.32
```

If you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6 Note:
For more information about SAND, please referring to http://ccl.cse.nd.edu/software/sand/.

4.6.2 Workload mpiBLAST

mpiBLAST is a parallel implementation of Basic Local Alignment Search Tool.
1 Required software stacks
MPICH2

2 Get workload mpiBLAST from BigDataBench
Download link for mpiBLAST http://www.mpiblast.org/Downloads/Stable
3 Prepare the input
The data set used by BFS is assembly of the human genome data. To get these data: http://prof.ict.ac.cn/bdb_uploads/Assembly_of_the_human_genome_data.tar.gz

4 Run the workload
Unpack the downloaded tar file

```
$tar -zxvf mpiBLAST-1.6.0.tgz
$cd mpiblast-1.6.0
```

Install mpiBLAST

```
$./configure
$make ncbi
$make
$make install
```

After this step, there will be three executable files **(mpiblast, mpiblast_cleanup, mpiformatdb)** under directory */usr/local/bin.*
3. Run the workload
All nodes **mount a shared direcotory**, we assume that the shared directory is **/mpiBlast/blastdb.**
Here, we assume that the mpiblast-1.6.0 is under **/mpiBLAST/mpiblast-1.6.0.**

```
$vim /root/.ncbirc
```

```
[NCBI]
Data=/mpiBLAST/mpiblast-1.6.0/ncbi/data
 [BLAST]
BLASTDB=/mpiBLAST/blast+/db
BLASTMAT=/mpiBLAST/mpiblast-1.6.0/ncbi/data
Shared=/mpiBLAST/blastdb
Local=/mpiBLAST/tmp
```

Save and exit /root/.ncbirc
Put the data to be alignmented under shared directory (/mpiBLAST/blastdb), we assume that the two data sets are **ref.fa** and **est.fa**.
Breaking data into several fragments, command is:

```
$cd /mpiBLAST/blastdb
$mpiformatdb -i ref.fa -pF –nfrags=NUMBER ### NUMBER is fragment number, suas as 4
$mpiformatdb -i est.fa -pF –nfrags=NUMBER
```

Run mpiBLAST, command is:

```
$mpirun -f machine_file -n PROCESS_NUM mpiblast -p blastn -d ref.fa -i est.fa -o result
```

5 Collect the running results
Note that if you want to collect the performance data on system or architecture level, you should run corresponding scripts in the background to collect data.
6 Note:
For more information about mpiBLAST, please referring to `http://www.mpiblast.org/Docs/Guide.`

4.7 Multitenancy

This tool focuses on a mix of workloads whose arrivals follow patterns hidden in real-world traces. Two type of representative data center workloads are considered:
- *Long-running service workloads.* These workloads offer online services such as web search engines and e-commerce sites to end users and the services usually keep running for months and years. The tenants of such workloads are service end users. - *Short-term data analytic workloads.* These workloads process input data of different scales (from KB to PB) using relatively short periods (from a few seconds to several hours). Example workloads are Hadoop, Spark and Shark jobs. The tenants of such workloads are job submitters.

4.7.1 Environment setup

1. Versions of software

CentOS 6.0
JKD 1.7
Python 2.7

2. Hadoop cluster setup

Refer to http://hadoop.apache.org/#Getting+Started

3. Environment setup of Nutch search engine

Refer to http://prof.ict.ac.cn/DCBenchmarks/Search_manual_v1.0.pdf
Search source code downloadïijŽhttp://prof.ict.ac.cn/DCBenchmarks
Note: In Search installation, if using normal user to login, you need to set password-free logins

4. Shark environment setup

Referred to https://github.com/amplab/shark/wiki/Running-Shark-on-a-Cluster

5. Environment variable configuration

Configure variables at /etc/profile
HADOOP_HOME=/opt/hadoop-1.2.1
SEARCH_H0ME=/opt/search/search

6. Copy the configuration file to $HADOOP_HOME/conf

```
$ cp randomwriter_conf.xsl workGenKeyValue_conf.xsl
$HADOOP_HOME/conf
```

4.7.2 Installation and Configuration of Software

Step 0: Download and unload the pakage of software

mixWorkloadSuite.tar at tmp form

Step 1: Prepare the input data

Compile Mapreduce job WriteToHdfs.java for writing input data set

```
$ cd /tmp/mixWorkloadSuite/FB
$ mkdir hdfsWrite
$ javac -classpath ${HADOOP_HOME}/hadoop-${HADOOP
_ VERSION}-core.jar -d hdfsWrite WriteToHdfs..java jar -cvf
WriteToHdfs.jar -C hdfsWrite/ .
```

Step 2: Edit *randomwriter_ conf.xsl* using configuration parameters

```
$ cd $HADOOP_ HOME//conf
$ vim randomwriter_ conf.xsl
```

Make sure the "test.randomwrite.bytes_ per_ map" and "java GenerateReplayScript" files have the same [size of each input partition in bytes] parameter.

Step 3: Execute the following commands

```
$bin/hadoop jar WriteToHdfs.jar org.apache.hadoop.examples.WriteToHdfs -
conf conf/randomwriter_ conf.xsl workGenInput
```

4.7.3 Generate the replay script

Step 0. Obtain a representative load

get-Job-Info.pl :This tool is used to analyze the default log format hadoop hadoop job history log data.
a. Instructions:

```
$ perl get-Job-Info.pl [job history dir] > outputFile.tsv
```

This script print to STDOUT, is used as a file into (outputFile.tsv) or further more in-depth analysis. This output file contents are divided by tap values (.tsv), the output file for each column as follows:
`1.unique_Job_id`
`2.submit_time_seconds`
`3.inter_job_submit_gap_seconds`
`4.map_input_bytes`
`5.shuffle_bytes`
`6.reduce_output_bytes`
Example of use:

```
$perl get-Job-Info.pl sort_LogRepository > outputFile.tsv
```

Description: sort Log Repository is the log file on the hadoop cluster running sort jobs directory on the local file system.
b. Import data [Workload trace processing] to give the log file [cleanup workload trace, extract the information needed]:

```
$FB-2009_samplesBySort_24_times_1hr_0.tsvoutputFile.tsv
| FB-2009_samples_24_times_1hr_0.tsv =>
FB-2009_samplesBySort_24_times_1hr_0.tsv
```

c. Use [matching] K-means clustering, a class of similar log file contains the contents of outputFile.tsv:
k_means_FB.py Use the format:

```
$python k_means_FB.py logFile.tsv K >
FB-2009_samplesKMSort_24_times_1hr_0.tsv
```

We find it N times the minimum loss value (K = 1,2, M) K = 10 to obtain the minimum time of the loss.
Example of use:

```
$python k_means_FB.py FB-2009_samplesBySort_24_times_1hr_0.tsv 10 >
FB-2009_samplesKMSort_24_times_1hr_0.tsv
```

Here, we provide a scripting tool run_clustering.sh and get_optimal_K.py to get the best K value. run_clustering.sh as follow:

```
$./ run_clustering.sh logFile.tsv [k Ranging from] [N Repetitions]
```

Example of use:

```
$./run_clustering.sh FB-2009_samplesBySort_24_times_1hr_0.tsv 1 20 50
```

Above script will generate/opt/mixWorkloadSuite/logfile/runlog_$k_$i+1.logfile, we use get_optimal_K.py. To analyze the /opt/mixWorkloadSuite/logfile/all files under optimal K value.

get_ optimal_ K.py

Example of use:

$python get_ optimal_ K.py /opt/mixWorkloadSuite/logfile/

d. Being the most representative of the load
After we get the last section 3.1.3 K clusters of log files, this stage needs to extract from this file contains a class file outputFile.tsv in content.
getTraceBySpecies_FB.py
Use the format

$python getTraceBySpecies_ FB.py FB-2009_ samplesKMSort _ 24_ times_ 1hr_ 0.tsv > FB-2009_ samplesKMBySort_ 24_ times_ 1hr_ 0.tsv

Step 1: Use GenerateReplayScriptFB.java to create a folder that includes the script of executable workload

```
$ cd /tmp/mixWorkloadSuite/FB
$ javac GenerateReplayScriptFB.java
$ java GenerateReplayScriptFB
      [Workload file]
      [Actual number of services generating clusters]
      [Number of testing clusters services from user ]
      [Input division size (byte)]
      [Input number of divisions]
      [Generated replay scripts catalog]
      [Inputted data directory on HDFS file system]
      [Workload output mark on HDFS file system]
      [Data amount of every reduce task]
      [workload standard error output directory ]
      [Hadoop command]
      [Directory of WorkGen.jar]
      [Directory of workGenKeyValue_conf.xsl ]
```

Workloadfile:

```
Workloadfile:
[path to synthetic workload file] for testing,
e.g FB-2009_samplesKMBySort_24_times_1hr_0.tsv
Actual number of services generating clusters:
[number of machines in the original production cluster]
Number of testing clusters services from user:
[number of machines in the cluster where the workload will be run]
Input division size (byte):
[size of each input partition in bytes] Should be roughly the same
as HDFS block size, e.g., 67108864
Input number of divisions:
[number of input partitions] The input data size need to be >= max
input size in the synthetic workload. Try a number. The program will
check whether it is large enough. e.g., 10 for the workload in
FB-2009_samplesKMBySort_24_times_1hr_0.tsv
Generated replay scripts catalog:
[output directory for the scripts] e.g., scriptsTestFB
Inputted data directory on HDFS file system:
[HDFS directory for the input data] e.g., workGenInput. Later,
need to generate data to this directory.
Workload output mark on HDFS file system:
[prefix to workload output in HDFS] e.g., workGenOutputTest. The
HDFS output dir will have format $prefix-$jobIndex.
Data amount of every reduce task::
[amount of data per reduce task in byptes] Should be roughly the
same as HDFS block size, e.g., 67108864
workload standard error output directory:
[workload output dir] Directory to output the log files, e.g.,
/home/USER/swimOutput.
Hadoop command:
[hadoop command] Command to invoke Hadoop on the targeted system,
e.g. $HADOOP_HOME/bin/hadoop
Directory of WorkGen.jar:
[path to WorkGen.jar] Path to WorkGen.jar on the targeted system,
e.g. $HADOOP_HOME/WorkGen.jar
Directory of workGenKeyValue_conf.xsl:
[path to workGenKeyValue_conf.xsl] Path to workGenKeyValue_conf.xsl
on the targeted system, e.g. $HADOOP_HOME/conf/workGenKeyValue_
conf.xsl
```

Step 2: Prepare replay scripts for Google workload traces

When use BigDataBench-multitenancy, we need to prepare scripts to workload replay. Here we use GenerateReplayScriptGoogle.java to generate the replay scripts

```
$ cd  /tmp/mixWorkloadSuite/Google
$ Javac GenerateReplayScriptGoogle.java
$ Java  GenerateReplayScriptGoogle
```

```
[workload file directory]
[replay scripts catalog]
[shark commad]
```

4.7.4 Workload replay in BigDataBench- multitenancy

Execute workload replay, just execute mixWorkloadReplay.sh using command line. Using method

```
$cd /tmp/mixWorkloadSuite/FB
$ cp -r scriptsTestFB $HADOOP_HOME
$ cd /tmp/mixWorkloadSuite/Google
$ cp -r scriptsTestGoogle $HADOOP_HOME
$ ./mixWorkloadReplay.sh argment(f/g or m)
```

4.8 Simulator Version

Simics is a full-system simulator used to run unchanged production binaries of the target hardware at high-performance speeds. It can simulate systems such as Alpha, x86-64, IA-64, ARM, MIPS (32- and 64-bit), MSP430, PowerPC (32- and 64-bit), POWER, SPARC-V8 and V9, and x86 CPUs.
We use SPARC as the instruction set architecture in our Simics version simulator benchmark suite, and deploy Solaris operation systems
1. Simics installation
It is recommended to install in the /opt/virtutech **directory**
Step 1. Download the appropriate Simics installation package from the download site, such as simics-pkg-00-3.0.0-linux.tar
Step 2. Extract the installation package, the command is as follows:

```
$ tar xf simics-pkg-00-3.0.0-linux.tar
```

It Will add a temporary installation directory, called simics-3.0-install
Step 3. Enter the temporary installation directory, run the install script, the command is as follows

```
$ cd simics-3.0-install $ sh install-simics.sh
```

Step 4. The Simics requires a decryption key, which has been unpacked before. decode key has been cached in $HOME/.simics-tfkeys.

```
$HOME/.simics-tfkeys
```

Step 5. When the installation script is finished, Simics has been installed in the /opt/virtutech/simics-<version>/, if the previous step to specify the installation path, this path will be different
2 Workloads
In the simulator version we provide the following workloads in our images, which is called BigDataBench Subset.

No.	Workload name
1	Hadoop-WordCount
2	Hadoop-Grep
3	Hadoop-NaiveBayes
4	Cloud-OLTP-Read
5	Hive-Differ
6	Hive-TPC-DS-query3
7	Spark-WordCount
8	Spark-Sort
9	Spark-Grep
10	Spark-Pagerank
11	Spark-Kmeans
12	Shark-Project
13	Shark-Orderby
14	Shark-TPC-DS-query8
15	Shark-TPC-DS-query10
16	Impala-Orderby
17	Impala-SelectQuery

3 Workloads running
Get Images Get Images from http://prof.ict.ac.cn/bdb_uploads/master.tar.gz and http://prof.ict.ac.cn/bdb_uploads/slaver.tar.gz
Decompress the packages.

```
$ tar -zxvf master.tar.gz
$ tar -zxvf slaver.tar.gz
```

Start the workloads
Users can use the following commands to drive the Simics images and start the workloads:
Hadoop Based workloads
Experimental environment
Cluster: one master one slaver,
Software : We have already provide the following software in our images.
Hadoop version: Hadoop-1.0.2

ZooKeeper version: ZooKeeper-3.4.5
Hbase version: HBase-0.94.5
Java version: Java-1.7.0

Running command

Workload	Master	Slaver
Wordcount	cd /master	cd /slaver
	./simics -c Hadoopwordcount_L	./simics -c Hadoopwordcount_L
	bin/hadoop jar $HADOOP_HOME/ hadoop-examples-*.jar wordcount /in /out/wordcount	
Grep	cd /master	cd /slaver
	./simics -c Hadoopgrep_L	./simics -c Hadoopgrep_LL
	bin/hadoop jar $HADOOP_HOME/ hadoop-examples-*.jar grep /in /out/g rep a*xyz	
NaiveBayes	cd /master	cd /slaver
	./simics -c HadoopBayes_L	./simics -c HadoopBayes_LL
	bin/mahout testclassifier -m /model -d /testdata	
Cloud OLTP-Read	cd /master	cd /slaver
	./simics -c YCSBRead_L	./simics -c YCSBRead_LL
	./bin/ycsb run hbase -P workloads/workloadc -p operationcount=1000 -p hosts=10.10.0.13 -p columnfamily=f1 -threads 2 -s>hbase_tranunlimited C1G.dat	

Hive based workloads
Experimental environment
Cluster: one master one slaver
Hadoop version: Hadoop-1.0.2
Hive version: Hive-0.9.0
Java version: Java-1.7.0

Running command

Workload	Master	Slaver
Hive-Differ	cd /master	cd /slaver
	./simics HiveDiffer_L	./simics -c HiveDiffer_LL
	./BigOP-e-commerce-difference.sh	
Hive-TPC-DS-query3	cd /master	cd /slaver
	./simics -c Hadoopgrep_L	./simics -c Hadoopgrep_LL
	./query3.sh	

Spark based version
Experimental environment
Cluster: one master one slaver
Hadoop version: Hadoop-1.0.2
Spark version: Spark-0.8.0
Scala version: Scala-2.9.3
Java version: Java-1.7.0

Running command

Workload	Master	Slaver
Spark-WordCount	cd /master	cd /slaver
	./simics -c SparkWordcount_L	./simics -c SparkWordcount_LL
	./run-bigdatabench cn.ac.ict.bigdatabench.WordCount spark://10.10.0.13:7077 /in /tmp/wordcount	
Spark-Grep	cd /master	cd /slaver
	./simics -c Sparkgrep_L	./simics -c Sparkgrep_LL
	./run-bigdatabench cn.ac.ict.bigdatabench.Grep spark://10.10.0.13:7077 /in lda_wiki1w /tmp/grep	
Spark-Sort	cd /master	cd /slaver
	./simics -c SparkSort_L	./simics -c SparkSort_LL
	./run-bigdatabench cn.ac.ict.bigdatabench.Sort spark://10.10.0.13:7077 /in /tmp/sort	
Spark-Pagerank	cd /master	cd /slaver
	./simics -c SparkPagerank_L	./simics -c SparkPagerank_LL
	./run-bigdatabench cn.ac.ict.bigdatabench.PageRank spark://10.10.0.13:7077 /Google_genGraph_5.txt 5 /tmp/PageRank	
Spark-Kmeans	cd /master ./simics -c SparkKmeans_L	cd /slaver ./simics -c SparkKmeans_LL
	./run-bigdatabench org.apache.spark.mllib.clustering.KMeans spark://10.10.0.13:7077 /data 8 4	

Shark based workloads Experimental environment
Cluster: one master one slaver
Software:
Hadoop version: Hadoop-1.0.2
Spark version: Spark-0.8.0
Scala version: Scala-2.9.3
Shark version: Shark-0.8.0

Hive version: hive-0.9.0-shark-0.8.0-bin
Java version: Java-1.7.0

Running command

Workload	Master	Slaver
Shark-Project Shark-Orderby	cd /master	cd /slaver
	./simics -c Sharkprojectorder_L	./simics -c Sharkprojectorder_LL
	./runMicroBenchmark.sh	
Shark-TPC-DS-query8	cd /master	cd /slaver
	./simics -c Sharkproquery8_L shark -f query8.sql	./simics -c Sharkquery8_LL
Shark-TPC-DS-query10	cd /master	cd /slaver
	./simics -c Sharkproquery10_L	./simics -c Sharkquery10_LL
	shark -f query10.sql	

4.9 Nutch Search Engine

4.9.1 Introduction

Search is a search engine model, which is used to evaluate datacenter and cloud computing systems.
Search v1.0 brings some simplicity in terms of installation, deployment and monitoring. Within this version, we are offering *Search* with everything inside and ready to go. *Search* consists of a search engine, a workload generator, and a comprehensive workload characterization tool—*DCAngel*.

i. Targeted Audience
This document is targeting two types of audiences:

- People who just want to use *Search* as a benchmark tool for evaluating their datacenter and cloud computing systems. This is for those who will directly use the provided *Search* benchmark directly to deploy it on their cluster.
- People who would like to modify the sources to fit their particular needs. You could use modified Search to do workloads characteristics analysis, add some functionality, or replace a component with another one.

ii. Structure of the document
This document goes on the following route:

- A detailed introduction will be given in 4.9.2, for people who have never used **Search** before.
- How to install *Search* version 1.0 is introduced in 4.9.3, for people who are not going to make any change to the provided *Search*.
- How to build an appliance on your own needs can be found in 4.9.4, for people who are going to modify some components of *Search*.

iii. Further Readings
The following links give more in-depth details about technologies used in *Search* v1.0.

- Nutch : http://nutch.apache.org
- Perf : https://perf.wiki.kernel.org/index.php/Main_Page
- Tomcat: http://tomcat.apache.org/
- Sqlite3: http://www.sqlite.org/
- Numpy: http://numpy.scipy.org/
- Matplotlib: http://matplotlib.sourceforge.net/

4.9.2 Search

i. Quick introduction
Search is a search engine site benchmark that implements the core functionality of a search engine site: providing indices and snapshot for a query term. It does not implement complementary services like crawling and ranking. It only has one kind of session — user's session, via which users can query terms. *Search* consists of three parts — a search engine, a workload generator and *DCAngel*.

The search engine is based on *nutch* which is an open source web-search software project. For *Search* v1.0, we use nutch-1.1 as the search engine's platform. The indices and snapshot we used in *Search* are generated by nutch-1.1 with SoGou Chinese corpus (http://www.sogou.com/labs/dl/t.html).

We get a real world search engine's trace from a user's log of SoGou (http://www.sogou.com/labs/dl/q.html). The workload generator can transform the real trace by specifying the query rate variation and terms' situation. The workload generator can also replay the real or synthetic traces.

DCAngel is a comprehensive workload characterization tool. It can collect performance metrics and then write them into database for further analysis and visualization. We use *perf* to collect performance counters' data.

For further reading about Search, please look at the following site: http://prof.ncic.ac.cn/DCBenchmarks.

ii. Available implementations
You may find available information and descriptions about older Search versions at its home page (http://prof.ncic.ac.cn/DCBenchmarks). If newer version implemented, it will be appended.

4.9.3 Getting started

In this part, you will drive right into the configuration and running part, supposing you don't want to modify the provided *Search*.

i. Overview

Our experiment platform is based on Nutch's distributed search engine which is a typical two-tier web application. It offers the following architecture:

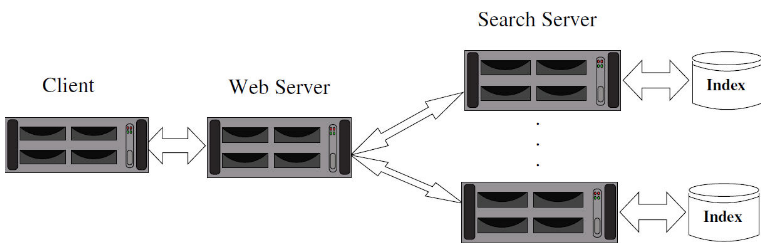


Fig. 1. Architecture of Search

- Client: injecting the workload thanks to the workload generator (written in python) and collecting metric results by DCAngel.
- Web Server: receiving HTTP requests from clients and dispatching them to Search Servers. We use Apache Tomcat 6.0.26 as the front end and nutch-1.1 as the search engine.
- Search Server: serving client requests transmitting by Web Server and the return the results to Web Server

ii. Prerequisites

The provided *Search v1.0* relies on *perf, JDK, Python and Numpy.* In this part, we focus on how you can use what is provided in the *Search-v1.0* package, for deeper information you may go over the Building part in 4.9.4.
Tomcat 6.0.26 and nutch-1.1 are included in our package, so the user should not prepare them.

ii.a. Linux Kernel Version

For this step, you need to get the root privileges for your Linux servers.
We need to build a linux kernel whose version is 2.6.31 or newer for all the **Search Server** nodes, because those kernels support *perf_ events* port, which is used by *perf.* When you compare the kernel, you should make sure that *perf_ events* is build into your kernel.

ii.b. perf

For *perf* , users should get a linux kernel source code whose version is 2.6.31 or newer on all **Search Server** nodes and then enter the directory *tools/perf.* After that, users should execute the following commands to install *perf*:

```
make
make install
```

ii.c. Python
All the linux systems need *Python* whose version is 2.7. Older or newer versions haven't been verified in our system.

ii.d. Numpy
The **Client** node needs *Numpy* (http://numpy.scipy.org/), which is the fundamental package needed for scientific computing with Python. You may need the following libraries or tools before installing *Numpy*:
atlas, python-nose, lapack, blas, libgfortran, python-dateutil, python-matplotlib, python-tz, python-setuptools

ii.e. Matplotlib
The **Client** node needs *matplotlib*(http://matplotlib.sourceforge.net/), which is a python 2D plotting library.

ii.f. JAVA
Java 1.6.x, preferably from Sun, must be installed in all linux systems except **Client node**. You should also set JAVA_HOME to the ans42 user.

ii.g. CPU
For this version, the **Search Server** nodes' CPU type must be as below:

1. Intel Xeon processor 3000, 3200, 5100, 5300 series
2. Intel Core 2 duo processor

If you use other CPUs, you may go over the CPU part in 4.9.4.

ii.h. SSH
SSH must be installed and *sshd* must be running. To run the *Search* scripts that manage remote daemons, please make sure that you can *ssh* on remote nodes without entering password

ii.i. Setup passphraseless ssh
Client node must *ssh* to **Web server and Search Server** nodes without a passphrase, Now check that.

```
$ ssh localhost
```

If you cannot ssh to nodes without a passphrase, execute the following commands at Client node:

```
$ ssh-keygen -t dsa -f $HOME/.ssh/id_dsa -P ""
This should result in two files, $HOME/.ssh/id_dsa (private key) and
$HOME/.ssh/id_dsa.pub (public key).
Copy $HOME/.ssh/id_dsa.pub to Web Server nodes and Search Server nodes
On those nodes run the following commands:
$ cat id_dsa.pub » $HOME/.ssh/authorized_keys2
$ chmod 0600 $HOME/.ssh/authorized_keys2
Depending on the version of OpenSSH the following commands may also be re-
quired:
$ cat id_dsa.pub » $HOME/.ssh/authorized_keys
$ chmod 0600 $HOME/.ssh/authorized_keys
An alternative is to create a link from authorized_keys2 to authorized_keys:
$ cd $HOME/.ssh && ln -s authorized_keys2 authorized_keys
On the Client node test the results by ssh'ing to other nodes:
$ ssh -i $HOME/.ssh/id_dsa server
```

This allows ssh access to the nodes without having to specify the path to the id_dsa file as an argument to ssh each time.

ii.j. Network

This should come as no surprise, but for the sake of completeness we have to point out that all the machines must be able to reach each other over the network. The easiest is to put all machines in the same network with regard to hardware and software configuration, for example connect machines via a single hub or switch and configure the network interfaces to use a common network such as 192.168.0.x/24.
To make it simple, we will access machines using their hostname, so you should write the IP address and the corresponding hostname into /etc/hosts. The following is an example.

```
#/etc/hosts
10.10.104.47 gd47
10.10.104.48 gd48
10.10.104.49 gd49
10.10.104.50 gd50
```

iii. Deploying Search

You're suggested creating a new user for all Linux systems, and use the new user to do the following. To make it simple, we just assume the new user you created for the tool is **ans42** with the password 'a'.
The user should download the *Search-v1.0* package to the **Client** node using the user *ans42*. We assume that you put the decompressed package in the directory of *$Search*. All the following operations should be done in **Client** node.

iii.a. Configuration

To deploy Search, you should first configure the $Search/common.mk file as follow.

```
uname = ans42 # the user's name for the benchmark
upwd = a # the corresponding password of the user
Master = gd88 # the Web Server node's hostname
Node = gd48,gd49,gd88 # the hostname of Web Server node and Search Server nodes
```

Do not change other configurations in this file.

At last, execute "**make deploy**" and "**source ∼/.bashrc**". Then Search will be deployed on all nodes. The deployment time depends on the number of nodes and the machine's hardware configuration. It maybe needs tens of minutes.

Before you running the benchmark, please make sure that the **Web Server** node's port 9090 is available or the **Web Server** node's firewall has already been closed.

iv. Running Benchmark

iv.a. Workload Preparation

Enter the *$Search/exp* directory and edit the run-test.sh file.

```
11 #———write your workload here————————#
12 report search.example.head:100000-fixed:100@s?i2@reqs-SoGou
```

Here, we give an example of workload at line 12, which is also a default workload. You can go over the workload part of session 4 if you want to create a new workload yourself.

If you want to use the default workload, you should replace the "?" by the number of Search Server nodes.

iv.b. Start benchmark test

Under the $Search/exp/ directory you should run the following command to start the benchmark test.

```
$ make test
```

The information of the test can be seen at file *./nohup.out*

iv.c. Get result

We have integrated *DCAngel*, which is a comprehensive workload characterization tool in our Search benchmark. Now we can use it to collect performance date, aggregate data and visualize data.

Figure.2 shows the high-level diagram of *DCAngel*. It stores performance data in a relational database managed by SQLite3 that supports the extended SQL statements. Users can access those data through the extended SQL statements.

All the tests' log and performance data collected by DCAngel can be find in the *$Search/exp/log/($workload)* directory. The *($workload)* here represents the workload you use. For example, if you use the default workload, the log can be find at *exp/log/search.example.head:100000-fixed:100@s?i2@reqs-SoGou* where "?" represents the Search server nodes' number. In that directory, there will be a file named *exp-report* if the test of the workload finished. The file is an empty file, and the only usage is to tell the user that workload replay has finished. The *exp-log* file records the start time and end time of the workload. The *search* directory collect the search log, the terms send to search engine and warm-up log. The hmon directory collects performance data of **Search Server** nodes.

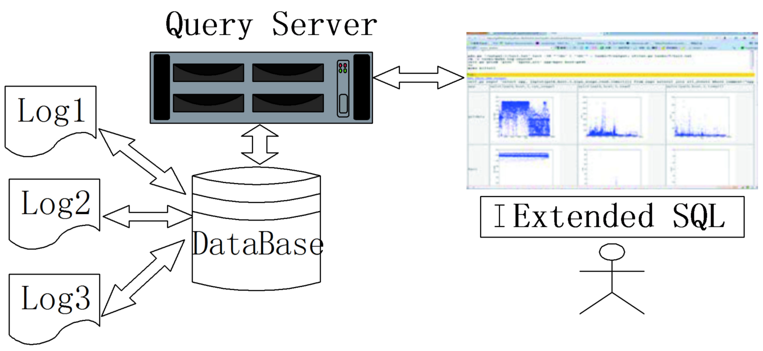
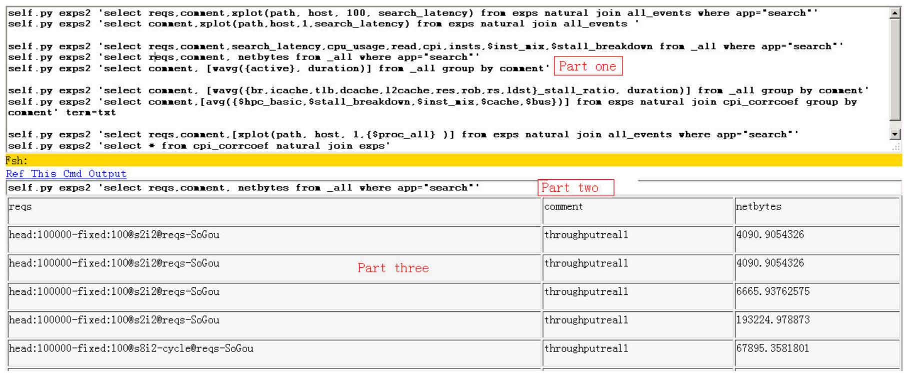


Fig. 2. High Level Diagram of DCAngel

Users can get data through a browser using *DCAngel*. For this version, the only browser we supported is *FireFox*. First, you should start the service by executing the following commands.

```
Enter the directory python-lib/fsh/:
$ cd python-lib/fsh
Start the service: ./psh.py port. For the port, we use 8002 as a example.
$./psh.py 8002
```

And then you can visit *DCAngel's* browser port through the address (do not forget the slash after "fsh"):
The $Search above is the location of Search-v1.0 package.
Figure 3 shows the snapshot of *DCAngel's* GUI. The GUI can be divided into three parts. Part one is commands column. Each line in that column is a *DCAngel* command. Users can execute the command by **ctrl+ left mouse button** click. Users can edit those commands to meet your requirement. Part two is command



```
self.py exps2 'select reqs,comment,xplot(path, host, 100, search_latency) from exps natural join all_events where app="search"'
self.py exps2 'select comment,xplot(path,host,1,search_latency) from exps natural join all_events '

self.py exps2 'select reqs,comment,search_latency,cpu_usage,read,cpi,insts,$inst_mix,$stall_breakdown from _all where app="search"'
self.py exps2 'select reqs,comment, netbytes from _all where app="search"'
self.py exps2 'select comment, [wavg({active}, duration)] from _all group by comment'
```

```
self.py exps2 'select comment, [wavg({br,icache,tlb,dcache,l2cache,res,rob,rs,ldst}_stall_ratio, duration)] from _all group by comment'
self.py exps2 'select comment,[avg({$hpc_basic,$stall_breakdown,$inst_mix,$cache,$bus})] from exps natural join cpi_corrcoef group by
comment' term=txt

self.py exps2 'select reqs,comment,[xplot(path, host, 1,{$proc_all} )] from exps natural join all_events where app="search"'
self.py exps2 'select * from cpi_corrcoef natural join exps'
Fsh:
```

```
self.py exps2 'select reqs,comment, netbytes from _all where app="search"'
```

reqs	comment	netbytes
head:100000-fixed:100@s2i2@reqs-SoGou	throughputreal1	4090.9054326
head:100000-fixed:100@s2i2@reqs-SoGou	throughputreal1	4090.9054326
head:100000-fixed:100@s2i2@reqs-SoGou	throughputreal1	6665.93762575
head:100000-fixed:100@s2i2@reqs-SoGou	throughputreal1	193224.978873
head:100000-fixed:100@s8i2-cycle@reqs-SoGou	throughputreal1	67895.3581801

Fig. 3. snapshot of DCAngel's GUI

input column; you can input your command here and execute it by pressing **Enter**. Part three is a display column, which displays the result of the command. Now we will show you the *DCAngel* command's grammar, so that you can writer your own commands.
A *DCAngel* command has two parts-a fixed part and a SQL like part. Let us look at the following command as an example.

self.py exps2 'select reqs,comment, netbytes from _all where app="search" '

The fixed part is self.py exps2 and the SQL like part is **'select reqs,comment, netbytes from _all where app="search" '**. For the SQL like part, users can write any statement that meets the sqlite3's syntax.
DCAngel's feedback may take a few seconds if it is your first time to execute a *DCAngel* command after a test. That is because *DCAngel* needs time to write metrics data it collected into database. *DCAngel* also defines many extend SQL functions. Those functions usage are shown as below.
std(arg1) : standard deviation of arg1
corrcoef(arg1, arg2) : correlation coefficient between arg1 and arg2
correlate(arg1,arg2) : cross correlation of arg1 and arg2
wavg(arg1,arg2): weighted average of arg1, and arg2 is weight
xplot(arg1, arg2, arg3, arg4) : draw the scatter figure of arg4. The x-axis of this figure is time and the y-axis is arg4's average value. arg1 and arg2 should be "path" and "host" respective. arg3 is degree of data aggregation. If arg3 equals 100, each point in the figure represents the average value of 100 arg4.
xhist(arg1, arg2, arg3, arg4) : draw the histogram of arg4's occurrence times. The x-axis of this figure is occurrence times and the y-axis is arg4's average value. arg1 and arg2 should be "path" and "host" respective. arg3 is degree of data aggregation. If arg3 equals 100, each value on the x-axis represents the average value of 100 arg4.

xscatter(arg1,arg2,arg3,arg4,arg5) : draw bi-dimensional histogram of arg4 and arg5. arg1 and arg2 should be "path" and "host" respective. arg3 is degree of data aggregation. If arg3 equals 100, each value on x-axis and y-axis represents the average value of 100 arg4 and arg5.
xcorr(arg1,arg2,arg3,arg4,arg5) : plot the cross correlation between arg4 and arg5. arg1 and arg2 should be âĂIJpathâĂİ and "host" respective. arg3 is degree of data aggregation.
If you want to use xplot you must make sure that the following read color words are not changed:
self.py exps2 ’select reqs,comment,host, xplot(path, host, 1, $metric) from exps natural join all_events
self.py exps2 ’select reqs,comment,host, xhist(path, host, 1, $metric) from exps natural join all_events
self.py exps2 ’select reqs,comment,host, xscatter(path, host, 1, $metric,$metic) from exps natural join all_events
self.py exps2 ’select reqs,comment,host, xcorr(path, host, 1, $metric,$metric) from exps natural join all_events
For $metric it can be any $metircs can be any field in Appendix B
We list the table structure of DCAngel’s database in Appendix A. Users can look up Appendix A and write your own *DCAngel* command

4.9.4 Building your own Search

If you want to build your own *Search*, this part will give some advices.

i. CPU
If your **Search Server** nodes do not own a CPU whose type is one of the types we mentioned in 4.9.3, you should modify line 167 to line 201 of file *$Search/hmon/hmon.py*.

```
167 kperf_events_map = ”’
168 CPU_CLK_UNHALTED.CORE 3c # cpu_cycles
169 CPU_CLK_UNHALTED.BUS 13c # bus cycles
170 INST_RETIRED.ANY c0 # insets
171 ITLB_MISS_RETIRED c9 # itlb_misses
172 DTLB_MISSES.ANY 108 # dtlb_misses
173 L1I_MISSES 81 # icache_misses
174 L1D_REPL f45 # dcache_misses
175 L2_LINES_IN.ANY f024 # l2cache_misses
176
177 PAGE_WALKS.CYCLES 20c # page_walks
178 CYCLES_L1I_MEM_STALLED 86 # icache_stalls
179
180 BR_INST_RETIRED.ANY c4 # br_insts
181 BR_INST_RETIRED.MISPRED c5 # br_misses
182
```

```
INST_RETIRED.LOADS 1c0 # load_insts
INST_RETIRED.STORES 2c0 # store_insts
INST_RETIRED.OTHER 4c0 # other_insts
SIMD_INST_RETIRED.ANY 1fc7 # simd_insts
FP_COMP_OPS_EXE 10 # fp_insts

RESOURCE_STALLS.ANY 1fdc # res_stalls
RESOURCE_STALLS.ROB_FULL 1dc # rob_stalls
RESOURCE_STALLS.RS_FULL 2dc # rs_stalls
RESOURCE_STALLS.LD_ST 4dc # ldst_stalls
RESOURCE_STALLS.FPCW 8dc # fpcw_stalls
RESOURCE_STALLS.BR_MISS_CLEAR 10dc # br_miss_stalls

BUS_TRANS_ANY e070 # bus_trans
BUS_DRDY_CLOCKS 2062 # bus_drdy
BUS_BNR_DRV 2061 # bus_bnr
BUS_TRANS_BRD e065 # bus_trans_brd
BUS_TRANS_RFO e066 # bus_trans_rfo
”’
```

You should go over your CPU's software design manual and change hexadecimal number above to the corresponding CPU event number.

ii. Make your search engine

For default *Search*, we just supply a SoGou corpus's snapshot and indices and all the **Search Server** nodes have the same indices and snapshot (it also called segments in **nutch**). Your can use your corpus's snapshot and indices. With your snapshot and indices, you can separate the snapshot and index them by using the *nutch* command — *merge* and *index*. You should put each part of snapshot and index into Search Server nodes' */home/ans42/crawl/combinations* directory. The default *Search* gives you an example of the indices and snapshot's layout in each Search Server node's directory: */home/ans42/crawl/combinations*. After that, you should modify the configuration file s?i2.cfg in Cline node's *$Search/nutch* where '?' represents the number of **Search Server** nodes. The content of that configuration file is as follows:

```
server-list=gd87 gd88 gd89 gd90
gd87-crawl-dir=01
gd88-crawl-dir=23
gd89-crawl-dir=45
gd90-crawl-dir=67
```

The first line represents the **Search Servers'** hostnames. From the second line, each defines the directory name of corresponding **Search Server** node's snapshot and index.

iii. Creating your own workload

4.9.3 mentions you can create your own workload, and this section will explains how to create a workload.

Now we will show how to create a workload by show the syntax and explaining a given workload's meaning. The given workload is as follows:

Now we will show how to create a workload by show the syntax and explaining a given workload's meaning. The given workload is as follows:

Syntax:
search.#anno.function1(:args)-function2(:args)@configfile@reqfile
An example:
search. instance.head:10000-poisson:20@s8i2@reqs-sogou

"search" means that a search engine is under evaluation. We use dot(.) to link different parts.

"#anno" is the annotation of this workload; in the example we use "instance" to indicate that this workload is an instance.

"function1(:args)-function2(:args)" indicates the functions we use to the real request sequence. "function1" and "function2" is transforming function's name. The function can be found at Appendix C. "args" is the function's parameters. we use "-" to link transforming functions. In the example "*head:10000*" means that we use head function in Appendix C, head function's parameter is "*10000*". "*poisson:20*" means that we use **poisson** function in Appendix C and its parameter is "20"

"*@configfile*" indicates the configuration file we used for **Search Server**. The configuration file is in **Client** node's *$Search/nutch* directory.. In the example "*@s8i2* " means that we use **s8i2**.cfg as **Search Server** nodes' configuration file where **s8i2**.cfg is in **Client** node's *$Search/nutch* directory.

"@reqfile" indicates the original request sequence we use. The request sequence file is in **Client** node's $Search/search-engine/data directory. Appendix D lists the request sequence we have provided, and users can use one of them or a new one. In the example, "*@reqs-sogou*" means that we use **sogou** request and the request file is *$Search/search-engine/data/reqs-sogou.*

You can use all the function in Appendix C to create your own workload, and adopt your own **Search Server** nodes' configuration file and request. For how to configure Search Server nodes you can consult 4.9.4

4.9.5 Appendix A - Metrics collected by DCAngel

variable	Definition
Metrics from performance counters	
cpu_cycles	Core cycles when core is not halted
bus_cycles	Bus cycles when core is not halted
insts	Retired instructions
itlb_misses	Retired instructions that missed the ITLB
dtlb_misses	Memory accesses that missed the DTLB
icache_misses	Instruction Fetch Unit misses
dcache_misses	L1 data cache misses
page_walks	Duration of page-walks in core cycles
icache_stalls	Cycles during which instruction fetches stalled
br_insts	Retired branch instructions
br_misses	Retired mispredicted branch instructions.
load_insts	Instructions retired, which contain a load
store_insts	Instructions retired, which contain a store
other_insts	Instructions retired, which no load or store operation
simd_insts	Retired Streaming SIMD instructions
fp_insts	Floating point computational micro-ops executed
res_stalls	Resource related stalls
rob_stalls	Cycles during which the reorder buffer full
rs_stalls	Cycles during which the reserve station full
ldst_stalls	Cycles during which the pipeline has exceeded load or store limit or waiting to commit all stores
fpcw_stalls	Cycles stalled due to floating-point unit control word writes
br_miss_stalls	Cycles stalled due to branch misprediction
bus_trans	All bus transactions
bus_drdy	Bus cycles when data is sent on the bus
bus_bnr	Number of Bus Not Ready signals asserted
bus_trans_brd	Burst read bus transactions
bus_trans_rfo	Read For Ownership bus transactions
Metrics from /proc filesystem	
usr	User mode CPU time
nice	The CPU time of processes whose nice value is negative
sys	Kernel mode CPU time
idle	Idle time
iowait	Iowait time
irq	Hard interrupt time

softirq	Soft interrupt time
intr	The times of interrupt happened
ctx	Context switch times
procs	Process number
running	The number of processes that is running
blocked	The number of processes that is blocked
mem_total	Total memory
free	Memory that is not used
buffers	Size memory in buffer cache
cached	Memory that cache used
swap_cached	Memory that once was swapped out, but still in the swapfile
active	Memory that has been used more recently
inactive	Memory that is not active
swap_total	Total amount of physical swap memory
swap_free	Total amount of free swap memory
pgin	The number of pages that paged in from disk
pgout	The number of pages that paged out to disk
pgfault	The number of page fault
pgmajfault	The number of major page faults
active_conn	TCP active connection
passive_conn	TCP passive connection
rbytes	Received bytes
rpackets	Received packets
rerrs	Received error packets number
rdrop	Number of packets dropped by native network adapter
sbytes	Bytes sent
spackets	Packets sent
serrs	Number of error packets sent
sdrop	Number of packets dropped by remote network adapter
read	Times of disk reads
read_merged	Times of disk merged reads
read_sectors	Times of sectors read
read_time	The total time disk read
write	Times of disk writes
write_merged	Times of merged disk writes
write_sectors	Times of sectors write
write_time	The total time of disk write

4.9.6 Appendix B - DCAngel database table structure

For the meaning of all following table's abbreviations, users can go over Appendix A.

Table exps

field	Definition
path	The test performance data's path under exp/ directory
app	User used application's name
comment	The comment when user used to specify a
reqs	Request name
duration	The test's duration
host	Node's host name

Table _all

Field	Definition
path	The test performance data's path under exp/ directory
host	Node's host name
insts	The mean value of instruction number
cpi	Cycles per instruction
br_miss_ratio	Branch miss ratio
br_stall_ratio	Branch stall ratio
icache_stall_ratio	Icache stall ratio
tlb_stall_ratio	TLB stall ratio
dcaceh_stall_ratio	Dcache stall ratio
l2cache_stall_ratio	L2 Cache stall ratio
res_stall_ratio	Resource related stall ratio
rob_stall_ratio	Reorder buffer stall ratio
rs_stall_ratio	Reserve station stall ratio
ldst_stall_ratio	Load and store stall ratio
fpcw_stall_ratio	Float point unit stall ratio
br_mix	Branch instruction ratio
load_mix	Load instruction ratio
store_mix	Store instruction ratio
ldst_mix	Load and store instruction ratio
simd_mix	SIMD instruction ratio
fp_mix	Float point instruction ratio
other_mix	Instructions that except load and store ratio
bus_util	Bus utilization
bus_d_util	bus_drdy ratioïijĹusers can find bus_drdy and all the following abbreviations' meaning in Appendix AïijĽ
bus_bnr_ratio	bus_bnr ratio
bus_brd_ratio	bus_brd ratio
bus_rfo_ratio	bus_rfo_ratio

cpu_usage	CPU utilization
search_latency	Average query latency
search_start	Test start time
duration	The test's duration
netbytes	rnetbytes+snetbytes
netpackets	rnetpacket+snetpacket

The meaning of following field is the same as it in Appendix A. So we will not explain them here.	
iowait	
ctx	
active	
pgfault	
pgmajfault	
active_conn	
passive_conn	
read	
write	
read_sectors	
write_sectors	

For table_all, we also define some macro which you can use to simplify your inputting.

For example you can write a *DCAngel* command self.py exps2 'select $prim from _all ', which has the same function with self.py exps2 'select app, comment, reqs, host from _all'

Macros and their definitions

macros	definition
$prim	app, comment, reqs, host
$hpc_basic	insts, cpi, br_miss_ratio
$stall_breakdown	br_stall_ratio, icache_stall_ratio, tlb_stall_ratio, dcache_stall_ratio, l2cache_stall_ratio, res_stall_ratio, rob_stall_ratio, rs_stall_ratio, ldst_stall_ratio, fpcw_stall_ratio
$inst_mix	br_mix, load_mix, store_mix, ldst_mix, simd_mix, fp_mix, other_mix
$cache	itlb_miss_ratio, dtlb_miss_ratio, icache_miss_ratio, dcache_miss_ratio, l2cache_miss_ratio
$bus	bus_util, bus_d_util, bus_bnr_ratio, bus_brd_ratio, bus_rfo_ratio
$proc_basic	cpu_usage, iowait, ctx, active, pgfault, pgmajfault
$net	active_conn, passive_conn, netbytes, netpackets,
$disk	read, write, read_sectors, write_sectors
$proc_selected	cpu_usage,iowait,ctx,active,pgmajfault,read_sectors
$hpc_all	$hpc_basic, $cache, $bus, $inst_mix
$proc_all	$proc_basic,$net,$disk

4.9.7 Appendix C- The workload transforming function

In the following table, we use *qs* and *ts* represent query sequence and time sequence respectively.

Function name	parameters	Definition
head	$Total: $start	Get qs and ts from the sequence number of $start, and the total entry number of qs and ts is $Total, e.g. search.#anno.head:100:0@cf@req If $start is 0 then is can be leaved out, e.g. search,#anno,head:100@cf@req
uniq	NULL	Get the unique query terms out of qs e.g. search.#anno.uniq@cf@req
random	$Total	Randomly get query terms from qs and the total number of queried terms is $Total,e.g. search.#anno.random:1000@cf@req
shuffle	NULL	Shuffle the terms in qs, e.g. search.#anno.shuffle@cf@req
hot	NULL	Sort the qs according to the frequency of terms' occurrence, e.g. search.#anno.hot@cf@req
lens	NULL	Sort the qs according to terms' length.
blockreq	$Blocksize: $repeatCount	Repeat every $Blocksize terms in qs $RepeatCount times. e.g. search.#anno.blockreq:10:2@cf@req
fixed	$Rate	Generate ts and set the query rate to be $Rate queries per second. e.g. search.#anno.fixed:20@cf@req
burst	$Rate:$K	Generate ts and let ts be i*$K*$K/$Rate, where i=1…len(qs) e.g. search.#anno.burst:20:2@cf@req
scale	$Rate	Compress or amplify original ts by setting the query rate to be $Rate queries per second. e.g. search.#anno.scale:20@cf@req
poisson	$Rate	Generate ts and make the query rate variation fit poisson distribution, and set the average rate to be $Rate queries per second, e.g. search.#anno.poisson:40@cf@req
ratestep	$Init:$step:$K	Generate ts and set the initial query rate to be $Init. The rate will increase for ($K-1) times. Each time it will increase the value of $step. Finally ,it will be stable at the rate of "$Init + $step * ($K-1)" e.g. search.#anno.ratestep:20:5:20@cf@req

4.9.8 Appendix D-Request sequence and their definitions

Request sequence name	Definition
warmup.reqs	A warmup request sequence for benchmark ramp-up
reqs-SoGou	A real world request sequence from SoGou search engine
reqs-Abc	A real world request sequence
reqs-Xyz	A real world request sequence
reqs-by-freqs-SoGou	Sorting reqs-SoGou according to request term's query frequency.
reqs-by-freqs-Abc	Sorting reqs-Abc according to request term's query frequency.
reqs-by-freqs-Xyz	Sorting reqs-Xyz according to request term's query frequency.
reqs-by-lens-SoGou	Sorting reqs-SoGou according to request term's length.
reqs-by-lens-Abc	Sorting reqs-Abc according to request term's length.
reqs-by-lens-Xyz	Sorting reqs-Xyz according to request term's length.