BigDataBench 5.0
User Manual

ICT, Chinese Academy of Sciences

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

1 Introduction .................................................................................................................. 5
  1.1 Context ....................................................................................................................... 5
  1.2 Environment ............................................................................................................... 7
  1.3 Format Specification ............................................................................................... 7

2 Overview of Software Packages and Workloads .......................................................... 9

3 Installation and Configuration of Software .................................................................. 10
  3.1 Setting up Hadoop .................................................................................................... 10
  3.2 Setting up Spark ........................................................................................................ 14
  3.3 Setting up MPI ........................................................................................................... 17
  3.4 Setting up Hive .......................................................................................................... 19
  3.5 Setting up hive-0.10.0-cdh4.2.0(for Impala) ............................................................. 21
  3.6 Setting up Impala ...................................................................................................... 23
  3.7 Setting up MySQL .................................................................................................... 27
  3.8 Setting up TensorFlow ............................................................................................. 28
  3.9 Setting up PyTorch .................................................................................................... 28

4 Workloads .................................................................................................................... 29
  4.1 BDGS ......................................................................................................................... 29
    4.1.1 Get BDGS ............................................................................................................ 29
    4.1.2 Compile BDGS ................................................................................................... 30
    4.1.3 Generate data ..................................................................................................... 30
  4.2 Micro Benchmarks .................................................................................................... 31
    4.2.1 Sort, Grep, WordCount, MD5 ............................................................................. 31
    4.2.2 Connected Component (CC) ................................................................................ 34
    4.2.3 RandSample ........................................................................................................ 36
    4.2.3 FFT ..................................................................................................................... 38
    4.2.4 Matrix Multiply .................................................................................................... 40
    4.2.5 Select Query, Aggregation Query, Join Query .................................................. 41
    4.2.6 Aggregation, Cross Product, Difference, Filter, OrderBy, Project, Union ........ 43
    4.2.7 Convolution ........................................................................................................ 46
    4.2.8 Fully Connected .................................................................................................. 47
    4.2.9 Relu, Sigmoid, Tanh ........................................................................................... 49
    4.2.10 MaxPooling, AvgPooling ................................................................................. 50
1 Introduction

1.1 Context

Architecture, system, data management, and AI or machine learning communities pay greater attention to innovative big data and AI algorithms, architecture, and systems. However, complexity, diversity, frequently changed workloads, and rapid evolution of big data and AI systems raise great challenges, as there is a lack of simple but elegant abstractions that facilitate understanding these most important classes of modern workloads. First, for the sake of conciseness, benchmarking scalability, portability cost, reproducibility, and better interpretation of performance data, we need understand what are the most time-consuming classes of unit of computation among big data and AI workloads. Second, for the sake of fairness, the benchmarks must include diversity of data and workloads. Third, for co-design of software and hardware, the benchmarks should be consistent across different communities; Moreover, we need simple but elegant abstractions that help achieve both efficiency and general-purpose.

We specify the common requirements of Big Data and AI workloads only algorithmically in a paper-and-pencil approach, reasonably divorced from individual implementations. We capture the differences and collaborations among IoT, edge, datacenter and HPC in handling Big Data and AI workloads. We consider each big data and AI workload as a pipeline of one or more classes of units of computation performed on initial or intermediate data inputs, each of which we call a data motif. For the first time, among a wide variety of big data and AI workloads, we identify eight data motifs (PACT’18 paper)— including Matrix, Sampling, Logic, Transform, Set, Graph, Sort and Statistic computation, each of which captures the common requirements of each class of unit of computation. Other than creating a new benchmark or proxy for every possible workload, we propose using data motif-based benchmarks—the combination of one or more data motifs—to represent diversity of big data and AI workloads.
We release an open-source and scalable big data and AI benchmark suite—BigDataBench 5.0—for IoT, Edge, Datacenter and HPC. The current version BigDataBench 5.0 provides 13 representative real-world data sets and 44 benchmarks. The benchmarks cover seven workload types including AI, online services, offline analytics, graph analytics, data warehouse, NoSQL, and streaming from three important application domains: Internet services (including search engines, social networks, e-commerce), recognition sciences, and medical sciences. Our benchmark suite includes micro benchmarks, each of which is a single data motif, components benchmarks, which consist of the data motif combinations, and end-to-end application benchmarks, which are the combinations of component benchmarks. Meanwhile, data sets have great impacts on workloads behaviors and running performance (our CGO’18 paper). Hence, data varieties are considered with the whole spectrum of data types including structured, semi-structured, and unstructured data. Currently, the included data sources are text, graph, table, and image data. Using real data sets as the seed, the data generators—BDGS—generate synthetic data by scaling the seed data while keeping the data characteristics of raw data.

Modern datacenter computer systems are widely deployed with mixed workloads to improve system utilization and save cost. However, the throughput of latency-critical workloads is dominated by their worst-case performance-tail latency. To model this important application scenario, we propose an end-to-end application benchmark—DCMix to generate mixed workloads whose latencies range from microseconds to minutes with four mixed execution modes.

Modern Internet services workloads are notoriously complex in terms of industry-scale architecture fueled with machine learning algorithms. As a joint work with Alibaba, we release an end-to-end application benchmark—E-commerce Search to mimic complex modern Internet services workloads.

To measure and rank high performance AI computer systems (HPC AI) or AI supercomputers, we also release an HPC AI benchmark suite (AI500),
consisting of micro benchmarks, each of which is a single data motif, and component benchmarks, e.g., resnet 50.

1.2 Environment

This document presents user manual information on BigDataBench 5.0 – including a brief introduction and the setting up guidelines of big data and AI software stacks, and operating guide of all workloads in BigDataBench 5.0. The information and specifications contained are for researchers who are interested in big data and AI benchmarking.

Note that the user manual information in the following passage are tested in the environment as follows.

Recommended browser: IE or Chrome.
Recommended OS: Centos 6.0 or later.
Libraries:
JDK 1.6 or later (Recommend version: jdk1.8.0_65)
C compiler, such as gcc, and C++ compiler, such as g++.

1.3 Format Specification

The following typographic conventions are used in this user manual:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold</td>
<td>Bold for emphasis.</td>
</tr>
<tr>
<td>Italic</td>
<td>Italic for fold and file names.</td>
</tr>
<tr>
<td>$command</td>
<td>$command for command lines.</td>
</tr>
<tr>
<td>Contents</td>
<td>Contents for contents in configuration files.</td>
</tr>
<tr>
<td>Courier font</td>
<td>Courier font for screen output.</td>
</tr>
<tr>
<td>Footnote</td>
<td>Some exception explanations are put in footnote.</td>
</tr>
</tbody>
</table>
## Overview of Software Packages and Workloads

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Benchmark Type</th>
<th>Workload Type</th>
<th>Dataset</th>
<th>Software stacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>Offline analytics</td>
<td>Wikipedia entries</td>
<td>Hadoop, Spark, Flink, MPI</td>
<td></td>
</tr>
<tr>
<td>Grep</td>
<td>Offline analytics</td>
<td>Wikipedia entries</td>
<td>Hadoop, Spark, Flink, MPI</td>
<td></td>
</tr>
<tr>
<td>WordCount</td>
<td>Offline analytics</td>
<td>Wikipedia entries</td>
<td>Hadoop, Spark, Flink, MPI</td>
<td></td>
</tr>
<tr>
<td>MD5</td>
<td>Offline analytics</td>
<td>Wikipedia entries</td>
<td>Hadoop, Spark, MPI</td>
<td></td>
</tr>
<tr>
<td>Connected Component</td>
<td>Micro Benchmark</td>
<td>Graph analytics</td>
<td>Facebook social network</td>
<td>Hadoop, Spark, Flink, MPI, GraphLab</td>
</tr>
<tr>
<td>RandSample</td>
<td>Offline analytics</td>
<td>Wikipedia entries</td>
<td>Hadoop, Spark, MPI</td>
<td></td>
</tr>
<tr>
<td>FFT</td>
<td>Offline analytics</td>
<td>Two-dimensional matrix</td>
<td>Hadoop, Spark, MPI</td>
<td></td>
</tr>
<tr>
<td>Matrix Multiply</td>
<td>Offline analytics</td>
<td>Two-dimensional matrix</td>
<td>Hadoop, Spark, MPI</td>
<td></td>
</tr>
<tr>
<td>Read</td>
<td>NoSQL</td>
<td>ProfSearch resumes</td>
<td>HBase, MongoDB</td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>NoSQL</td>
<td>ProfSearch resumes</td>
<td>HBase, MongoDB</td>
<td></td>
</tr>
<tr>
<td>Scan</td>
<td>NoSQL</td>
<td>ProfSearch resumes</td>
<td>HBase, MongoDB</td>
<td></td>
</tr>
<tr>
<td>OrderBy</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
<td></td>
</tr>
<tr>
<td>Aggregation</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
<td></td>
</tr>
<tr>
<td>Project</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
<td></td>
</tr>
<tr>
<td>Filter</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
<td></td>
</tr>
<tr>
<td>Select</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
<td></td>
</tr>
<tr>
<td>Union</td>
<td>Data warehouse</td>
<td>E-commerce transaction</td>
<td>Hive, Spark-SQL, Impala</td>
<td></td>
</tr>
<tr>
<td>Convolution</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Pthread, PyTorch</td>
<td></td>
</tr>
<tr>
<td>Fully Connected</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Pthread, PyTorch</td>
<td></td>
</tr>
<tr>
<td>Relu</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Pthread, PyTorch</td>
<td></td>
</tr>
<tr>
<td>Sigmoid</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Pthread, PyTorch</td>
<td></td>
</tr>
<tr>
<td>Tanh</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Pthread, PyTorch</td>
<td></td>
</tr>
<tr>
<td>MaxPooling</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Pthread, PyTorch</td>
<td></td>
</tr>
<tr>
<td>AvgPooling</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Pthread, PyTorch</td>
<td></td>
</tr>
<tr>
<td>CosineNorm</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Pthread, PyTorch</td>
<td></td>
</tr>
<tr>
<td>BatchNorm</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Pthread, PyTorch</td>
<td></td>
</tr>
<tr>
<td>Dropout</td>
<td>AI</td>
<td>Cifar, ImageNet</td>
<td>TensorFlow, Pthread, PyTorch</td>
<td></td>
</tr>
</tbody>
</table>
### 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.

**1) Prerequisites**

Java JDK: version 1.6 or later (Recommend version: jdk1.8.0_65)
Hadoop version: we recommend version 2.7.1, which was used and tested in our environment.

2) Download Hadoop

Download Hadoop 2.7.1 from the following link:

3) Basic Configuration

Step 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:

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

This command will generate 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:

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

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

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

Step 2. Configure Hadoop

Decompress the Hadoop package.

```bash
$ tar -zxvf hadoop-2.7.1.tar.gz
```

Edit the configuration file:

```bash
$ cd hadoop-2.7.1/etc/hadoop
```

Add the following contents in “hadoop-env.sh”:

```bash
export JAVA_HOME=/path/to/java_home
```
Add the following contents in “core-site.xml”:

```xml
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/path/to/tmp_data</value>
    <description>Abase for other temporary directories.</description>
  </property>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://master_node_hostname:9100</value>
  </property>
</configuration>
```

Add the following contents in “hdfs-site.xml”:

```xml
<configuration>
  <property>
    <name>dfs.name.dir</name>
    <value>/path/to/store/metadata</value>
  </property>
  <property>
    <name>dfs.data.dir</name>
    <value>/path/to/store/hdfs_data</value>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

Add the following contents in “mapred-site.xml”:

```xml
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>master_node_hostname:9200</value>
  </property>
</configuration>
```
Add the following contents in “yarn-site.xml”

```xml
<configuration>
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>hostname</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.resource.cpu-vcores</name>
        <value>core_number</value>
    </property>
    <property>
        <name>yarn.scheduler.minimum-allocation-mb</name>
        <value>min_mb</value>
    </property>
    <property>
        <name>yarn.scheduler.maximum-allocation-mb</name>
        <value>max_mb</value>
    </property>
    <property>
        <name>yarn.nodemanager.resource.memory-mb</name>
        <value>res_md</value>
    </property>
</configuration>
```
Add slave hostname/IP in “slaves” file

hostname1
hostname2

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

$ vim ~/.bashrc

Add:

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

$ source ~/.bashrc

Then scp all this package (i.e., hadoop-2.7.1) to all slave nodes and put them under the same directory.

3) Start Hadoop

Step 1. Format the HDFS:

```bash
$ cd hadoop-2.7.1
$ bin/hadoop namenode -format
```

Step 2. Start Hadoop:

```bash
$ sbin/start-all.sh
```

4) Stop Hadoop:

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

3.2 Setting up Spark

1) Prerequisites

- Java JDK: version 1.6 or later
- Scala: version 2.10.4 or later
- Hadoop: version 2.7.1 or other 1.x/2.x version
- Spark: recommend version 1.5.2, which are tested in our environment, or other 1.x version
2) **Download Spark**

Download the prebuild package:

https://archive.apache.org/dist/spark/spark-1.5.2/spark-1.5.2-bin-hadoop2.6.tgz

3) **Basic Configuration**

**Step 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 command will generate 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. Run the following commands on worker nodes:

```
$ cat id_dsa.pub » $HOME/.ssh/authorized_keys
$ chmod 600 $HOME/.ssh/authorized_keys
```

On the master node test the results through ssh to worker nodes:

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

**Step 2. Configure Spark**

Decompress the Spark package.

```
$ tar -zxvf spark-1.5.2-bin-hadoop2.6.tgz
```

Edit the configuration file:

```
$ cd spark-1.5.2-bin-hadoop2.6/conf
$ cp spark-env.sh.template spark-env.sh
```

Edit `spark-env.sh`

```
SPARK_MASTER_IP= MASTER_HOSTNAME
export SCALA_HOME=/usr/lib/scala-2.10.4
export JAVA_HOME=/usr/java/jdk1.8.0_65
export HADOOP_HOME=/path/to/hadoop-2.7.1
```
export HADOOP_CONF_DIR=/path/to/hadoop-2.7.1/etc/hadoop
export SPARK_EXECUTOR_INSTANCES=instance_num
export SPARK_EXECUTOR_CORES=core_num
export SPARK_EXECUTOR_MEMORY=xxG
export SPARK_DRIVER_MEMORY=xxG
$ cp spark-defaults.conf.template spark-defaults.conf
Edit spark-defaults.conf

spark.master                     spark:// MASTER_HOSTNAME:7077
spark.eventLog.enabled           true
spark.default.parallelism 100
spark.storage.memoryFraction 0.4
spark.shuffle.memoryFraction 0.6
spark.shuffle.manager hash
spark.shuffle.compress true
spark.broadcast.compress true
spark.shuffle.file.buffer 64k
spark.storage.unrollFraction 0.5
spark.serializer org.apache.spark.serializer.KryoSerializer
spark.rdd.compress true

Edit slaves:

WORKER_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

3) Start Spark

$ cd spark-1.5.2-bin-hadoop2.6/
$ sbin/start-all.sh

4) Stop Spark

$sbin/stop-all.sh
3.3 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 your own installation, you can also choose a higher version.

1) Prerequisites

a C compiler, such as gcc.
a C++ compiler, such as g++

2) 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/

3) Basic Installation

Step 1. Unpack the tar file

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

Step 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:

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

Step 3. Build

Build command:

$ make 2>$1 | tee m.txt

Step 4. Install

Install command:
Step 5. Add the bin subdirectory to the PATH environment variable

For shell of bash and sh, using the command:

```bash
vim ~/.bashrc
```

```
export PATH=$PATH:/home/mpich2-ins/bin
```

Save and exit vim.

```bash
source ~/.bashrc
```

4) Check

Step 1. Checking the path

Using the command to display the path to your bin subdirectory:

```bash
which mpicc
which mpic++
```

In our example, the first command should display “/home/mpich2-ins/bin/mpicc”.

Step 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.

5) Use MPICH2 to run programs

Step 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 2. Compile an example C program using mpicc

Using the command to produce corresponding executable file:

```bash
mpicc cpi.c -o cpi
```

This command will produce an executable file named cpi

Step 3. Run the program using multiple processes on one or more machines
Using the command to run the program on local machine:

```bash
$ 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:

```bash
$ 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.

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

Note: -f parameter specifies the machine information, and -n parameter specifies 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

### 3.4 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.

#### 1) Prerequisites

Java JDK: version 1.6 or later

Hadoop: we recommend version 2.7.1, which was used and tested in our environment.

#### 2) Download and Install Hive
Step 1. Download the most recent stable release of Hive
We recommend version 1.2.1 (http://archive.apache.org/dist/hive/hive-1.2.1/), which was used and tested in our environment.

Step 2. Unpack the tarball
$ tar -xzf hive-1.2.1.tar.gz

Step 3. Set environment variable
HIVE_HOME(/path/to/hive-1.2.1), add $HIVE_HOME/bin to your PATH.

$ vim ~/.bash_profile
Edit the ~/.bash_profile:

   export HIVE_HOME=/path/to/hive-1.2.1
   export PATH=$HIVE_HOME/bin:$PATH

3) Basic Configuration

Step 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

Edit hive-env.sh:

   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

Step 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

4) Start Hive

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

   $ HIVE_HOME/bin/hive

5) 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;
```

6) Stop Hive

In Hive CLI, Type the following command:

```
$hive > exit;
```

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

1) Prerequisites

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

2) Download and Install hive-0.10.0-cdh4.2.0

Step 1. Download the hive-cdh from cloudera website.
We recommend hive-0.10.0-cdh4.2.0, which was tested in our environment.
http://archive.cloudera.com/cdh4/cdh/4/hive-0.10.0-cdh4.2.0.tar.gz

Step 2. Unpack the tarball.

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

Step 3. Set environment variable

HIVE_HOME (/path/to/hive-0.10.0-cdh4.2.0), add HIVE_HOME to your PATH.

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

Edit the ~/.bashrc file and add:

```
export HIVE_HOME=/path/to/hive-0.10.0-cdh4.2.0
export PATH=$HIVE_HOME/bin:$PATH
```

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

3) Basic Configuration

Step 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
```


Edit hive-env.sh and add:
HADOOP_HOME=$HADOOP_HOME
export HIVE_CONF_DIR=$HIVE_HOME/conf
export HIVE_AUX_JARS_PATH=$HIVE_HOME/lib

$ source hive-env.sh

Step 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

Step 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;

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

$ sudo vim $HIVE_HOME/conf/hive-site.xml
Edit core-site.xml:

```xml
<!DOCTYPE ?xml version="1.0"?>
<!DOCTYPE stylesheet type="text/xsl" href="configuration.xsl"?-->
<configuration>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://localhost:3306/metastore?createDatabaseIfNotExists=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>
</configuration>
```
<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>hive</value>
</property>

<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>hadoophive</value>
</property>

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

4) Start Hive-metastore and Hive-server
   $ HIVE_HOME/bin/hive

5) 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;

6) Stop Hive
   In Hive CLI, Type the following command:
   $ hive > exit;

3.6 Setting up Impala

1) 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 specific linux version. The details can be found in official document, which are

2) Download and install Impala

Step 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

Step 2. Download bigtop-utils-0.4+300-1.cdh4.0.1.p0.1.el6.noarch.rpm
http://archive.cloudera.com/impala/redhat/6/x86_64/impala/1/RPMS/noarch/

Step 3. Download libevent-1.4.13-4.el6.x86_64.rpm
http://rpm.pbone.net/index.php3?stat=26&dist=79&size=67428&name=libevent-1.4.13-4.el6.x86_64.rpm

Step 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

3) Basic Configuration

Step 1. Copy configuration files
Copy hive-site.xml, core-site.xml and hdfs-site.xml to the default directory of configuration directory /etc/impala/conf.

$ cp $HIVE_HOME/conf/hive-site.xml /etc/impala/conf/hive-site.xml
$ 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

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

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

<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>

Step 3. In all impala nodes, configure /etc/impala/conf/core-site.xml.

Edit core-site.xml:

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

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

<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.

Step 4. In all impala node, configure /etc/impala/conf/hdfs-site.xml.

Edit hdfs-site.xml,

<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>
<value>10000</value>
</property>

Step 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_BACKEND_PORT}
   use_statestore
   state_store_host=${IMPALA_STATESTORE_HOST}
   be_port=${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
   hive_home=/usr/lib/impala/sbin
   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
```

4) **Start Impala**

$ sudo service impala-state-store restart
$ sudo service impala-server restart
5) **Test Impala**

Use the follow command to view the start status of impala.

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

Enter the impala shell client.

```bash
$ 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...

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

```bash
[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 df844fb967c8740f8d3527ef)
```

6) **Stop Impala**

In Impala CLI, Type the following command:

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

### 3.7 Setting up MySQL

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

1) **Prerequisites**

CentOS: 6.5

2) **Install MySQL**

   **Step 1: Install Mysql by YUM**

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

   **Step 2: Initialize Mysql service**

   The information of initialization will be printed on the screen:

   ```bash
   $ sudo /usr/bin/mysql_secure_installation
   ```
Enter current password for root (enter for none): press "Enter" 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!

3) Test Mysql service

Step 1. Enter Mysql CLI.
$ mysql -uroot -phadoophive

Step 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;

3.8 Setting up TensorFlow

1) Prerequisites

python, pip, numpy, scipy

2) Download and install TensorFlow

We recommend TensorFlow 1.1.0 version.
$ pip install --upgrade
https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-1.1.0-cp27-none-linux_x86_64.whl

3) Test TensorFlow

$ python
>>>import tensorflow

If command “import tensorflow” doesn’t return errors, then TensorFlow is successfully installed.

3.9 Setting up PyTorch

We recommend PyTorch 1.0.1
1) Prerequisites

python, pip, numpy, scipy

2) Download and install PyTorch

pip install with Python 2.x version:
  pip3 install torch torchvision

pip install with Python 3.x version:
  pip install torch torchvision

conda install:
  conda install pytorch torchvision -c pytorch

conda install with specific cuda version:
  conda install pytorch torchvision cudatoolkit=10.0 -c pytorch

Install from source:
  Follow instructions at this URL: https://github.com/pytorch/pytorch#from-source

3) Test PyTorch

$ python
>>> import torch

If command “import torch” doesn’t return errors, then PyTorch is successfully installed.

4 Workloads

4.1 BDGS

BigDataBench 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 be tested, BDGS generates synthetic data sets that can be used as inputs of application specific workloads.

In the release edition, BDGS consist of three parts: Text generator, Graph generator, and Table generator.

4.1.1 Get BDGS

The BDGS has been packaged in each the benchmark suite, users do not need to download separately. User can download it from the following link:
Also, users can execute the obtain BDGS in each benchmark directory. Such as in http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_Hadoop.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:

1) Pre-required software
The BDGS depends gsl, if the systems do not have the package installed.
- $ wget http://prof.ict.ac.cn/bdb_uploads/bdb_4/BigDataGeneratorSuite.tar.gz
- $ cd BigDataGeneratorSuite

2) Compile Text data generate
Cd to the directory, install gsl and execute make command:
- $ cd BigDataGeneratorSuite/Text_datagen
- $ tar -xf gsl-1.15.tar.gz
- $ cd gsl-1.15 & ./configure & make & make install
- $ cd ..
- $ make

3) 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 BigDataGeneratorSuite/Graph_datagen again:
- $ cd ../
- $ make

4) 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 or prepare the data and the second step is to run the applications.

4.2 Micro Benchmarks

Data motifs are fundamental concepts and units of computation among a majority of big data and AI workloads. We design a suite of micro benchmarks, each of which is a single data motif implementation.

4.2.1 Sort, Grep, WordCount, MD5

The instructions of Sort, Grep, WordCount and MD5 workloads are similar, so we put them together. Here we use sort as an example, Grep, WordCount and MD5 running processes are the same, you can just change the “Sort” to “Grep” or “WordCount” or “MD5” in the following commands.

1) Hadoop based

Required Software Stacks Hadoop and BGDS

Step 1. Get workloads from BigDataBench
Download the benchmark package from http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 2. Decompress the Hadoop package.
$ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz
Install gsl:
$ cd BigDataBench_V5.0_BigData_MicroBenchmark/
$ ./prepar.sh

Step 3. Prepare the input
$ cd Hadoop/Sort
$ ./genData-sort.sh <size>
The parameter size means the input data size (GB)
You can find the generated text data in hdfs://hadoop/terasort/terasort-{size}G

Step 4. Run the workload
$ ./run-terasort.sh <size>
The parameter size means the input data size (GB)

Step 5. Collect the running results
The output of the workload will be put in hdfs with location /hadoop/terasort/terasort-out.

2) Spark based

Step 1. Required Software
Spark stack and BGDS

Step 2. Get workloads from BigDataBench
Download the benchmark package from http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 3. Decompress the Spark package.
   \$ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 4. Prepare the input
   \$ cd BigDataBench_V5.0_BigData_MicroBenchmark/
   \$ cd Spark/Sort
   \$ ./genData-Sort.sh <size>
   The parameter size means the input data size (GB)
   You can find the generated text data in hdfs: /spark/sort/sort-{size}G

Step 5. Run the workload
   \$ ./runSpark-Sort.sh <size>
   The parameter size means the input data size (GB)

Step 6. Collect the running results
   The output of the workload will be put in hdfs with location /spark/sort/output.

3) Flink based
Required Software Stacks Flink and BGDS

Step 1. Get workloads from BigDataBench
Download the benchmark package from http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 2. Decompress the Hadoop package.
   \$ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Install gsl:
   \$ cd BigDataBench_V5.0_BigData_MicroBenchmark/
   \$ ./prepar.sh

Step 3. Prepare the input
   \$ cd Flink/sort-grep-wc
   \$./genData_MicroBenchmarks.sh
   Then you need to input the data size (GB) you want to generate – “print data size GB”. The data will be generated in /data-MicroBenchmarks directory on HDFS.

Step 4. Run the workload
   \$ ./run_Microbenchmarks.sh <workload>
   The workload can be “Sort/Grep/Wordcount”.

Step 5. Collect the running results
   The output of the workload will be put in hdfs with location /flink-xxx-result.
4) **MPI based**

**Step 1. Required software stacks**
MPICH2

**Step 2. Get MPI workload from BigDataBench**
Download from the link:  
http://prof.ict.ac.cn/bdb uploads/bdb_5/packages/BigDataBench_V5.0_BigData MicroBenchmark.tar.gz

**Step 3. Prepare the input**
The data sets used by these four workloads are generated by big data generation tool (BDGS).
To generate data:

i) Unpack the downloaded tar file

```bash
$ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz
```

ii) Generate data for Sort:

```bash
$ cd BigDataBench_V5.0_BigData_MicroBenchmark/MPI/MPI_Sort
$ ./genData_Sort.sh
```

iii) Generate data for Grep:

```bash
$ cd BigDataBench_V5.0_BigData_MicroBenchmark/MPI/MPI_Grep
$ sh genData_grep.sh
```

iv) Generate data for WordCount:

```bash
$ cd BigDataBench_V5.0_BigData_MicroBenchmark/MPI/MPI_WordCount
$ sh genData_wordcount.sh
```

v) Generate data for MD5:

```bash
$ cd BigDataBench_V5.0_BigData_MicroBenchmark/MPI/MPI_MD5
$ sh genData_md5.sh
```

Input the data size you want to generate with the units of GB, such as 10 if you want to generate 10 GB data. After this step, it will generate data under respective directory.

**Step 4. Run the workload**

i) Install workload

```bash
$ cd BigDataBench_V5.0_BigData_MicroBenchmark/
$ cd MPI/MPI_Sort/(Grep/WordCount/MD5)
$ make
```

After this step, there will be one executable files named mpi_sort(/grep/wordcount/md5) under the current directory. Then you can run the workload.

ii) For Sort, command is:

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

iii) For Grep, command is:

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

iv) For WordCount, command is:

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

iv) For MD5, command is:

```bash
$mpirun -f machine_file -n PROCESS_NUM ./mpi_md5 input_file output_file
```

For example, the three command would be:

```bash
$mpirun -f machine_file -n 12 ./mpi_sort data-sort/in output
```
Step 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.

Note: For grep workload, the second parameter (pattern) in running command line means the expression needs to be matched.

4.2.2 Connected Component (CC)

1) Hadoop based

Required Software Stacks Hadoop and BGDS

Step 1. Get workloads from BigDataBench
Download the benchmark package from http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 2. Decompress the Hadoop package.

$ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_MicroBenchmark

Step 3. Prepare the input

$ cd Hadoop/CC
$ ./genData-cc.sh <log_vertex>

The parameter log_vertex indicates the vertex of the generated data, means vertex = 2^log_vertex.

You can find the generated graph data is under the hdfs directory: /hadoop/cc

Step 4. Run the workload

$ ./run-cc.sh <log_vertex>

The parameter log_vertex indicates the vertex of the generated data, means vertex = 2^log_vertex.

Step 5. Collect the running results

The output of the workload will be put in hdfs: concmpt_curbm, concmpt_tempbm, concmpt_nextbm, concmpt_output.

2) Spark based

Step 1. Required Software
Spark stack and BGDS

Step 2. Get workloads from BigDataBench
Download the benchmark package from http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 3. Decompress the Spark package.
Step 4. Prepare the input
$ cd Spark/CC
$./genData-cc.sh <log_vertex>
The parameter log_vertex indicates the vertex of the generated data, means vertex = 2^log_vertex.
You can find the generated graph data is under the hdfs directory: /spark/cc

Step 5. Run the workload
$./runSpark-cc.sh <log_vertex>
The parameter log_vertex indicates the vertex of the generated data, means vertex = 2^log_vertex.

Step 6. 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.

3) **MPI based**

Step 1. Required software stacks
MPICH2
Cmake: Cmake 2.8.12.2 is preferred
Boost 1.43.0
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_V5.0_MPI/MicroBenchmark/GraphAnalytics
$ cd ConnectedComponent/parallel-bgl-0.7.0
$ cmake .
$ cd parallel-bgl-0.7.0/libs/graph_parallel/test
$ make distributed_page_rank_test

Step 2. Get MPI workload from BigDataBench
Download from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 3. Prepare the input
The data set used by CC is generated by big data generation tool (BDGS).
To generate data:
i) Unpack the downloaded tar file
$ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

ii) Generate data for CC:
$ cd BigDataBench_V5.0_BigData_MicroBenchmark/MPIMPI_Connect
$ ./genData_connectedComponents.sh
Then you will be asked how many data you like to generate:
Please Enter The Iterations of GenGragh: (enter a number here, It means the number of vertices generated, represented by power of 2)
You can find the generated graph data: data-Connected_Components/Face book_genGragh_$I.txt ($I here is the number you entered).

Step 4. Run the workload
Run through linux command:
```
$ mpirun -f machine_file -n PROCESS_NUM ./run_connectedComponents InputGraphfile num_ofVertex num_ofEdges
```
Note that you can find the num_ofVertex and num_ofEdges information from the output of the data generating command.

Step 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.

4.2.3 RandSample

1) Hadoop based
Required Software Stacks Hadoop and BGDS

Step 1. Get workloads from BigDataBench
Download the benchmark package from
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 2. Decompress the Hadoop package.
```
$ tar -zxf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz
```
Install gsl:
```
$ cd BigDataBench_V5.0_BigData_MicroBenchmark/
$ ./prepar.sh
```

Step 3. Prepare the input
```
$ cd Hadoop/randSample
$./genData-randSample.sh <size>
```
The parameter size means the input data size (GB).
You can find the generated test data in hdfs:/hadoop/randsample/GB-randsampleHP

Step 4. Run the workload
```
$ ./run-randSample.sh <size> <sample_ratio>
```
Parameter size: the input data size, GB
Parameter sample_ratio: the sampling ratio, ranges from 0 to 1.

Step 5. Collect the running results
The output of the workload will be put in hdfs with location /hadoop/randsample/randsampleHP-result.

2) Spark based

Step 1. Required Software
Spark stack and BGDS

Step 2. Get workloads from BigDataBench
Download the benchmark package from
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 3. Decompress the Spark package.
   $ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 4. Prepare the input
   $ cd BigDataBench_V5.0_BigData_MicroBenchmark/
   $ cd Spark/randSample
   $ ./genData-randSample.sh <size>

The parameter size means the input data size (GB).
You can find the generated text data in hdfs: /spark/randSample/

Step 5. Run the workload
   $ ./runSpark-randSample.sh <size> <sample_ratio>
Parameter size: the input data size, GB
Parameter sample_ratio: the sampling ratio, ranges from 0 to 1.

Step 6. Collect the running results
The output of the workload will be put in hdfs with location /spark/randSample/output.

3) MPI based

Step 1. Required software stacks
MPICH2

Step 2. Get MPI workload from BigDataBench
Download from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 3. Prepare the input
The data sets used by these four workloads are generated by big data generation tool (BDGS).
To generate data:
   i) Unpack the downloaded tar file
      $ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz
   ii) Generate data for Sort:
       $ cd BigDataBench_V5.0_BigData_MicroBenchmark/MPI/mpiRandSample
       $ sh genData_randsample.sh
       Input the data size you want to generate with the units of GB, such as 10 if you want to generate 10 GB data. After this step, it will generate data under RandSample directory.

Step 4. Run the workload
   i) Install workload
$ cd BigDataBench_V5.0_BigData_MicroBenchmark/MPI/mpiRandSample
$ make
After this step, there will be one executable files named mpi-randsample under the current directory. Then you can run the workload.

ii) Run the workload

$mpirun -f machine_file -n PROCESS_NUM ./mpi-randsample input_file output_file

**Step 5. Collect the running results**

When the workload run is complete, it will generate the output_file.

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.

### 4.2.3 FFT

1) **Hadoop based**

Required Software Stacks Hadoop and BGDS

**Step 1. Get workloads from BigDataBench**

Download the benchmark package from

http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_Micro_Benchmark.tar.gz

**Step 2. Decompress the Hadoop package.**

$ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Install gsl:

```
$ cd BigDataBench_V5.0_BigData_MicroBenchmark/
$ ./prepar.sh
```

**Step 3. Prepare the input**

```
$ cd Hadoop/FFT
$ ./genData-fft.sh <row_num> <col_num> <sparsity>
```

“sparsity” ranges from 0 to 1, which means the ratios that the matrix elements are zero. 0 represents no element is zero while 1 represents all elements are zero.

After the run process, it will generate the input data on HDFS under directory: /hadoop/fft/

**Step 4. Run the workload**

```
$ ./run-fft.sh <row_num> <col_num> <sparsity>
```

The parameters are the same with the generating command.

**Step 5. Collect the running results**

The output of the workload will be put in hdfs with location /hadoop/fft/fft-result.

2) **Spark based**

**Step 1. Required Software**

Spark stack and BGDS

**Step 2. Get workloads from BigDataBench**

Download the benchmark package from
Step 3. Decompress the Spark package.

$ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 4. Prepare the input

$ cd BigDataBench_V5.0_BigData_MicroBenchmark/
$ cd Spark/FFT
$ ./genData-fft.sh <row_num> <col_num> <sparsity>

“sparsity” ranges from 0 to 1, which means the ratios that the matrix elements are zero. 0 represents no element is zero while 1 represents all elements are zero.

After the run process, it will generate the input data on HDFS under directory:
/spark/fft/

Step 5. Run the workload

$ ./runSpark-fft.sh <row_num> <col_num> <sparsity>

The parameters are the same with the generating command.

Step 6. Collect the running results

The output of the workload will be put in hdfs with location /spark/fft/output.

3) MPI based

Step 1. Required software stacks
MPICH2

Step 2. Get MPI workload from BigDataBench
Download from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 3. Prepare the input

The data sets used by these four workloads are generated by big data generation tool (BDGS).

To generate data:

i) Unpack the downloaded tar file

$ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

ii) Generate data for fft:

$ cd BigDataBench_V5.0_BigData_MicroBenchmark/MPI/mpiFFT
$ sh genData-fft.sh <row_num> <col_num> <sparsity>

“sparsity” ranges from 0 to 1, which means the ratios that the matrix elements are zero. 0 represents no element is zero while 1 represents all elements are zero.

After the run process, it will generate the input data: genData-Matrix/fft-data

Step 4. Run the workload

i) Install workload

$ cd BigDataBench_V5.0_BigData_MicroBenchmark/MPI/mpiFFT
$ make

After this step, there will be one executable files named mpi fft under the current directory. Then you can run the workload.
ii) Run the workload

```
$mpirun -f machine_file -n PROCESS_NUM ./mpifft genData-Matrix/fft-data
```

**Step 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.

### 4.2.4 Matrix Multiply

1) Hadoop based

Required Software Stacks Hadoop and BGDS

**Step 1. Get workloads from BigDataBench**

Download the benchmark package from [http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz](http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz)

**Step 2. Decompress the Hadoop package.**

```
$ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz
```

**Step 3. Prepare the input**

```
$ cd BigDataBench_V5.0_BigData_MicroBenchmark/Hadoop/MatrixMult

$./genData-matMult.sh <sparsity> <row_i> <col_i> <col_j>
```

sparsity: the percentage of zero elements, ranges from 0 to 1.
row_i: the row number of matrix A
col_i: the column number of matrix A
col_j: the column number of matrix B

After the run process, it will generate the input data on HDFS under directory: /hadoop/matMult/mat1 and /hadoop/matMult/mat2

**Step 4. Run the workload**

```
$ ./run-matMult.sh <sparsity> <row_i> <col_i> <col_j>
```

sparsity: the percentage of zero elements, ranges from 0 to 1.
row_i: the row number of matrix A
col_i: the column number of matrix A
col_j: the column number of matrix B

**Step 5. Collect the running results**

The output of the workload will be put in hdfs with location /hadoop/matMult/mat-out.

2) Spark based

**Step 1. Required Software**

Spark stack and BGDS

**Step 2. Get workloads from BigDataBench**

Download the benchmark package from [http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz](http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz)

**Step 3. Decompress the Spark package.**
Step 4. Run the workload

The workload use random generate dataset, so we don’t need to generate data by ourselves.

$ ./runSpark_matrixMult.sh <row_i> <col_i> <col_j>

row_i: the row number of matrix A
col_i: the column number of matrix A
col_j: the column number of matrix B

Step 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.

4.2.5 Select Query, Aggregation Query, Join Query

1) Hive based

Step 1. Required Software Stacks

Java JDK: version 1.6 or later
Hadoop: we recommend version 2.7.1, which was used and tested in our environment.
Hive: we recommend version 1.2.1, which was used and tested in our environment.

Step 2. Get workloads from BigDataBench

Download the benchmark package from

http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 3. Prepare the input

Make sure Hadoop and Hive have been successfully started.
Unpack the downloaded tar file:

$ tar -zxvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_MicroBenchmark
$ cd Hive/Interactive_Query
$ ./gen_data.sh

Step 4. Run the workloads;

$ ./run_AnalyticWorkload.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.

Step 5. Collect the running results

When the workload run 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

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>


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

---

**2) Impala version**

**Step 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

**Step 2. Get workloads from BigDataBench**
Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

**Step 3. Prepare the input**

Make sure Hadoop, Hive and Impala have been successfully started. Here we use Aggregation workload as an example, the others are the same under respective directory.

Unpack the downloaded tar file:

```bash
$ tar -xzvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_MicroBenchmark
```

Then execute `gen_data.sh`:

```bash
$ ./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.
```

**Step 4. Run the workload**

Modify `impala_restart.sh`, replace your impala node with actual impala node ip. For example,

```
for i in localhost
```

Run the workloads:

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

**Step 5. Collect the running results**

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

```
Logging initialized using configuration in file:/home/cdh4/hive-0.10.0-cdh4.2.0/
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/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
```

**4.2.6 Aggregation, Cross Product, Difference, Filter, OrderBy, Project, Union**

**1) Hive version**

**Step 1. Required Software Stacks**

Java JDK: version 1.6 or later
Hadoop: we recommend version 2.7.1, which was used and tested in our environment.

Hive: we recommend version 1.2.1, which was used and tested in our environment.

**Step 2. Get workloads from BigDataBench**
Download the benchmark package from

http://prof.ict.ac.cn/bdbUploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

**Step 3. Prepare the input**
Make sure Hadoop and Hive have been successfully started.

Unpack the downloaded tar file:

```bash
$ tar -xzf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_MicroBenchmark
$ cd Hive/Interactive_MicroBenchmark
$ ./gen_data.sh
```

**Step 4. Run the workload**

```bash
$ ./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.

**Step 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:
```
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>


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/warehouse
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

2) Impala version

Step 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

Step 2. Get workloads from BigDataBench
Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz

Step 3. Prepare the input
Make sure Hadoop, Hive and Impala have been successfully started.
Unpack the downloaded tar file:
$ tar -xzvf BigDataBench_V5.0_BigData_MicroBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_MicroBenchmark
$ cd Impala/Interactive_MicroBenchmark
Step 4. Run the workload

Modify impala_restart.sh, replace your impala node with actual impala node ip.
For example,

```bash
for i in localhost
```

Run the workloads;
```bash
$ ./run_MicroBenchmark.sh
```

Step 5. Collect the running results

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

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 bindings found 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

4.2.7 Convolution

1) TensorFlow based

Step 1. Required Software Stacks
Python 2 or 3
Scipy and Numpy
TensorFlow 1.0+

Step 2. Get workloads from BigDataBench

Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_Micro
Benchmark.tar.gz

Step 3. Prepare the input

The TensorFlow micro benchmarks use random generate data, you need to specify
the *batch size, image size, channel, and filter size.*

Step 4. Run the workloads;

```bash
$ tar –xf BigDataBench_V5.0_AI_MicroBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_MicroBenchmark/TensorFlow
```
$ python conv2d.py <batch_size> <img_size> <channel> <filter_size>
Running conv2d with scripts:
  $ ./run-tensorflow.sh conv <datasize>
Parameter “datasize” can be large/medium/small.
large: 224*224*64 (means length, width and channel respectively)
medium: 112*112*128
small: 56*56*256

Step 5. Collect the running results
When the workload run complete, it will display the running time information.

2) Pthreads based

Step 1. Required Software Stacks
  g++ compiler
  OpenCV, recommend 3.2 version
  Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2

Step 2. Get workloads from BigDataBench
Download the Benchmark from this link:
  http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_MicroBenchmark.tar.gz

Step 3. Prepare the input
  $ cd BigDataBench_V5.0_AI_MicroBenchmark/Pthread
  The Pthread micro benchmarks use ImageNet as data input, ImageData directory contains the image data with three sizes: Image_1000, Image_10000 and Image_100000.

Step 4. Compile the workload
  $ make
  This command will produce an executable file named conv2d.

Step 5. Run the workload
  $ ./conv2d ./ImageData/image_Simgsize/imgSimgsize/ NCHW 12 227 227 100
  Here $simgsize can be 1000, 10000, or 100000.

Step 6. Collect the running results
When the workload run is complete, it will display the output.

4.2.8 Fully Connected

1) TensorFlow based

Step 1. Required Software Stacks
  Python 2 or 3
  Scipy and Numpy
  TensorFlow 1.0+
Step 2. Get workloads from BigDataBench
Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_MicroBenchmark.tar.gz

Step 3. Prepare the input
The TensorFlow micro benchmarks use random generate data, you need to specify the batch size, image size, and channel.

Step 4. Run the workloads;
$ tar –xf BigDataBench_V5.0_AI_MicroBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_MicroBenchmark/TensorFlow
$ python matmul.py <batch_size> <img_size> <channel>
Running fully connected with scripts:
$ ./run-tensorflow.sh matmul <datasize>
Parameter “datasize” can be large/medium/small.
large: 224*224*64 (means length, width and channel respectively)
medium: 112*112*128
small: 56*56*256

Step 5. Collect the running results
When the workload run complete, it will display the running time information.

2) Pthreads based

Step 1. Required Software Stacks
g++ compiler
OpenCV, recommend 3.2 version
Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2

Step 2. Get workloads from BigDataBench
Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_MicroBenchmark.tar.gz

Step 3. Prepare the input
$ cd BigDataBench_V5.0_AI_MicroBenchmark/Pthread
The Pthread micro benchmarks use ImageNet as data input, ImageData directory contains the image data with three sizes: Image_1000, Image_10000 and Image_100000.

Step 4. Compile the workload
$ make
This command will produce an executable file named matmul.

Step 5. Run the workload
$ ./matmul ../ImageData/image_$imgsize/img$imgsize/ 12 227 227 100
Here $imgsize can be 1000, 10000, or 100000.
Step 6. Collect the running results

When the workload run is complete, it will display the output.

4.2.9  Relu, Sigmoid, Tanh

We use the Relu as an example, the running processes of Sigmoid and Tanh are the same, you can just change the relu in the command to sigmoid or tanh.

1) TensorFlow based

Step 1. Required Software Stacks

Python 2 or 3
Scipy and Numpy
TensorFlow 1.0+

Step 2. Get workloads from BigDataBench

Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AIMicroBenchmark.tar.gz

Step 3. Prepare the input

The TensorFlow micro benchmarks use random generate data, you need to specify the batch size, image size, and channel.

Step 4. Run the workloads:

$ tar –xf BigDataBench_V5.0_AIMicroBenchmark.tar.gz
$ cd BigDataBench_V5.0_AIMicroBenchmark/TensorFlow
$ python relu.py <batch size> <img size> <channel>

Running relu/sigmoid/tanh with scripts:

$ ./run-tensorflow.sh <workload> <datasize>

Parameter “workload” can be relu/sigmoid/tanh.
Parameter “datasize” can be large/medium/small.
large: 224*224*64 (means length, width and channel respectively)
medium: 112*112*128
small: 56*56*256

Step 5. Collect the running results

When the workload run complete, it will display the running time information.

2) Pthreads based

Step 1. Required Software Stacks

g++ compiler
OpenCV, recommend 3.2 version
Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2

Step 2. Get workloads from BigDataBench

Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_MicroBenchmark.tar.gz

**Step 3. Prepare the input**

```bash
$ cd BigDataBench_V5.0_AI_MicroBenchmark/Pthread
```


**Step 4. Compile the workload**

```bash
$ make
```

This command will produce an executable file named relu, sigmoid or tanh.

**Step 5. Run the workload**

```bash
$ ./relu ./.ImageData/image_$imgsize/img$imgsize/ 12 227 227 100
```

Here `$imgsize` can be 1000, 10000, or 100000.

**Step 6. Collect the running results**

When the workload run is complete, it will display the output.

### 4.2.10 MaxPooling, AvgPooling

We use the MaxPooling as an example, the running process of AvgPooling is the same, you can just change the `max_pool` in the command to `avg_pool`.

1) **TensorFlow based**

**Step 1. Required Software Stacks**

- Python 2 or 3
- Scipy and Numpy
- TensorFlow 1.0+

**Step 2. Get workloads from BigDataBench**

Download the Benchmark from the link:

http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_MicroBenchmark.tar.gz

**Step 3. Prepare the input**

The TensorFlow micro benchmarks use random generate data, you need to specify the `batch size`, `image size`, and `channel`.

**Step 4. Run the workloads;**

```bash
$ tar --xf BigDataBench_V5.0_AI_MicroBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_MicroBenchmark/TensorFlow
$ python max_pool.py <batch_size> <img_size> <channel>
```

Running MaxPooling and AvgPooling with scripts:

```bash
$ ./run-tensorflow.sh <workload> <datasize>
```

Parameter “workload” can be maxpool or avgpool.
Parameter “datasize” can be large/medium/small.
large: 224*224*64 (means length, width and channel respectively)
Step 5. Collect the running results
When the workload run complete, it will display the running time information.

2) Pthreads based

Step 1. Required Software Stacks
- g++ compiler
- OpenCV, recommend 3.2 version
- Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2

Step 2. Get workloads from BigDataBench
Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_MicroBenchmark.tar.gz

Step 3. Prepare the input
- $ cd BigDataBench_V5.0_AI_MicroBenchmark/Pthread

The Pthread micro benchmarks use ImageNet as data input, ImageData directory contains the image data with three sizes: Image_1000, Image_10000 and Image_100000.

Step 4. Compile the workload
- $ make

This command will produce an executable file named max_pool or avg_pool.

Step 5. Run the workload
- $ ./max_pool ../ImageData/image_$imgsize/img$imgsize/ 12 227 227 100
Here $imgsize can be 1000, 10000, or 100000.

Step 6. Collect the running results
When the workload run is complete, it will display the output.

4.2.11 CosineNorm, BatchNorm
We use the BatchNorm as an example, the running process of CosineNorm is the same, you can just change the batch_norm in the command to cosine_norm.

1) TensorFlow based

Step 1. Required Software Stacks
- Python 2 or 3
- Scipy and Numpy
- TensorFlow 1.0+

Step 2. Get workloads from BigDataBench
Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_MicroBenchmark.tar.gz

**Step 3. Prepare the input**

The TensorFlow micro benchmarks use random generate data, you need to specify the *batch size, image size*, and *channel*.

**Step 4. Run the workloads**;

```bash
$ tar –xf BigDataBench_V5.0_AI_MicroBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_MicroBenchmark/TensorFlow
$ python batch_normalization.py <batch_size> <img_size> <channel>
```

Running batchNorm with scripts:

```bash
$ ./run-tensorflow.sh batchNorm <datasize>
```

Parameter “datasize” can be large/medium/small.
- large: 224*224*64 (means length, width and channel respectively)
- medium: 112*112*128
- small: 56*56*256

**Step 5. Collect the running results**

When the workload run complete, it will display the running time information.

---

2) *Pthreads based*

**Step 1. Required Software Stacks**

g++ compiler

OpenCV, recommend 3.2 version

Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2

**Step 2. Get workloads from BigDataBench**

Download the Benchmark from this link:

http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_MicroBenchmark.tar.gz

**Step 3. Prepare the input**

```bash
$ cd BigDataBench_V5.0_AI_MicroBenchmark/Pthread
```


**Step 4. Compile the workload**

```bash
$ make
```

This command will produce an executable file named batch_norm.

**Step 5. Run the workload**

```bash
$ ./batch_norm ../ImageData/image_$imgsize/img$imgsize/ 12 227 227 100
```

Here $imgsize can be 1000, 10000, or 100000.

**Step 6. Collect the running results**

When the workload run is complete, it will display the output.
4.2.12 Dropout

1) TensorFlow based

Step 1. Required Software Stacks
- Python 2 or 3
- Scipy and Numpy
- TensorFlow 1.0+

Step 2. Get workloads from BigDataBench
Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_MicroBenchmark.tar.gz

Step 3. Prepare the input
The TensorFlow micro benchmarks use random generate data, you need to specify the batch size, image size, and channel.

Step 4. Run the workloads;
$ tar --xf BigDataBench_V5.0_AI_MicroBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_MicroBenchmark/TensorFlow
$ python dropout.py <batch_size> <img_size> <channel>
Running fully connected with scripts:
$ ./run-tensorflow.sh dropout <datasize>
Parameter “datasize” can be large/medium/small.
large: 224*224*64 (means length, width and channel respectively)
medium: 112*112*128
small: 56*56*256

Step 5. Collect the running results
When the workload run complete, it will display the running time information.

2) Pthreads based

Step 1. Required Software Stacks
- g++ compiler
- OpenCV, recommend 3.2 version
- Dependences: libopencv_core.so.3.2 and libopencv_imgproc.so.3.2

Step 2. Get workloads from BigDataBench
Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_MicroBenchmark.tar.gz

Step 3. Prepare the input
$ cd BigDataBench_V5.0_AI_MicroBenchmark/Pthread

**Step 4. Compile the workload**

```
$ make
```

This command will produce an executable file named dropout.

**Step 5. Run the workload**

```
$ ./dropout ../ImageData/image_$imgsize/img$imgsize/ 12 227 227 100
```

Here $imgsize can be 1000, 10000, or 100000.

**Step 6. Collect the running results**

When the workload run is complete, it will display the output.

### 4.3 Component Benchmarks

Considering the benchmarking scalability, we use the motif combinations to compose original complex workloads with a DAG-like structure considering the data pipeline. The DAG-like structure is to use a node representing original or intermediate data set being processed, and an edge representing a data motif.

#### 4.3.1 Image Classification

**1) TensorFlow based**

**Step 1. Required Software Stacks**

1. tensorflow-gpu 1.12 or tensorflow 1.12
   
   ```
   $ pip install tensorflow-gpu==1.12 or pip install tensorflow==1.12
   ```

   if you want to build tensorflow from source, see
   [https://www.tensorflow.org/install/source](https://www.tensorflow.org/install/source)

2. Cuda 9.0


3. Cudnn 7.4.2

   Downloads cudnn 7.4.2: [https://developer.nvidia.com/cudnn](https://developer.nvidia.com/cudnn)

   Installation guide for linux: [https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html](https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html)

**Step 2. Get workloads from BigDataBench**

Download the Benchmark from the link:
[http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz](http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz)

```
$ tar –xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd TensorFlow/Image_classification
```
Step 3. Prepare the input

2. Convert these raw images to TFRecords by using build_imagenet_data.py script.

Step 4. Run the workloads

```python
$python imagenet_main.py --data_dir=/path/to/imagenet
```

Both the training dataset and the validation dataset are in the same directory. The model will begin training and will automatically evaluate itself on the validation data roughly once per epoch.

Some running options:

```
--model_dir: to choose where to store the model
--resnet_size: to choose the model size (options include ResNet-18 through ResNet-200)
--num-gpus: to choose computing device
   0: Use OneDeviceStrategy and train on CPU
   1: Use OneDeviceStrategy and train on GPU
   2+: Use Mirroredstrategy (data parallelism) to distribute a batch between devices
```

Full list of options, see resnet_run_loop.py

Step 5. Collect the running results

When the workload run is complete, it will display the output.

2) PyTorch based

Step 1. Required Software Stacks

1. Python 2.7
2. Anaconda 5.3.0
   ```
curl -O https://repo.anaconda.com/archive/Anaconda2-5.3.0-Linux-x86_64.sh
sh Anaconda2-5.3.0-Linux-x86_64.sh
   ```
3. Pytorch 1.0
   ```
conda install pytorch torchvision cudatoolkit=9.0 -c pytorch
( https://pytorch.org/get-started/locally/ )
   ```

Step 2. Get workloads from BigDataBen

Download the Benchmark from this link:
[http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz](http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz)

```bash
$ tar -xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
```

Step 3. Prepare the input

1. Download the ImageNet dataset
2. Move validation images to labeled subfolders, you can use the following script:
Step 4. Run the workload
bash run_image_classify ${batchSize} ${dataDir}

Step 5. Collect the running results
When the workload run is complete, it will display the output.

4.3.2 Image Generation

1) TensorFlow based

Step 1. Required Software Stacks
python 2.7
tensorflow >= 1.2 (verified on 1.2 and 1.3)
tqdm

Step 2. Get workloads from BigDataBench
Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb/uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ tar –xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd TensorFlow/Image_generation

Step 3. Prepare the input
$ python download.py lsun
$ python convert.py

Step 4. Run the workloads;
$ python train.py --help

optional arguments:
  -h, --help        show this help message and exit
  --num_epochs NUM_EPOCHS, default: 20
  --batch_size BATCH_SIZE, default: 128
  --num_threads NUM_THREADS # of data read threads (default: 4)
  --model MODEL (DCGAN, LSGAN, WGAN, WGAN-GP, EBGAN, BEGAN, DRAGAN, CoulombGAN)
  --name NAME, default: name=model
  --dataset DATASET, -D DATASET (CelebA / LSUN)
  --ckpt_step CKPT_STEP # of steps for saving checkpoint (default: 5000)
  --renew train model from scratch-clean

Step 5. Collect the running results
When the workload run is complete, it will display the output.
2) PyTorch based

Step 1. Required Software Stacks
PyTorch
PyTorchvision

Step 2. Get workloads from BigDataBench
Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz

$ tar -xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd PyTorch/Image_generation

Step 3. Prepare the input
You can download the dataset by:
$ python3 lsun/download.py -o <data_dir> -e bedroom
Or you can also download the lsun dataset from http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet

Step 4. Run the workload
$ cd WGAN
$ python main.py --mlp_G --ngf 512 --dataset lsun --dataroot <lsun-train-folder> --cuda

Step 5. Collect the running results
When the workload run is complete, it will display the output.

4.3.3 Text-to-Text Translation

1) TensorFlow based

Step 1. Required Software Stacks
tensorflow-gpu
tensorflow

Step 2. Get workloads from BigDataBench
Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz

$ tar -xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd TensorFlow/Text_to_Text

Step 3. Prepare the input
Download tensor2tensor from https://github.com/tensorflow/tensor2tensor. And make sure you can access to the Internet. For compatibility you need to change the io file of python.

```
```

Change:
```
    def rename(oldname, newname, overwrite=False):
```

To:
```
    def rename(oldname, newname, overwrite=True):
```

Step 4. Run the workloads;
```
    $./run.sh
```

Step 5. Collect the running results
When the workload run is complete, it will display the output.

### 2) PyTorch based

Step 1. Required Software Stacks
PyTorch 1.0.1

Step 2. Get workloads from BigDataBench
Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz

```
    $ tar –xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
    $ cd BigDataBench_V5.0_AI_ComponentBenchmark/
    $ cd PyTorch/Text_to_Text
```

Step 3. Prepare the input
```
    $ sh download.sh
```
Or you can also find the dataset from:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet

Step 4. Run the workload
```
    $ sh run.sh
```

Step 5. Collect the running results
When the workload run is complete, it will display the output.
```
    [Info] Finished.
```

### 4.3.4 Image to Text

#### 1) TensorFlow based

Step 1. Required Software Stacks
Bazel
Step 2. Get workloads from BigDataBench

Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0.AI_ComponentBenchmark.tar.gz

$ tar –xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd TensorFlow/Image_to_Text

Step 3. Prepare the input

You can download the dataset via COCO webpage, or http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet.

After downloading the dataset, please execute the following command.

$ IM2TXT_HOME=/path/to/your/coco2014-dataset
$ # Directory containing preprocessed MSCOCO data.
$ MSCOCO_DIR="${IM2TXT_HOME}/im2txt/data/mscoco"
$ # Inception v3 checkpoint file.
$ INCEPTION_CHECKPOINT="${IM2TXT_HOME}/im2txt/data/inception_v3.ckpt"
$ # Directory to save the model.
$ MODEL_DIR="${IM2TXT_HOME}/im2txt/model"
$ # Build the model.
$ cd research/im2txt
$ bazel build -c opt //im2txt/..

Step 4. Run the workloads;

$ bazel-bin/im2txt/train
   --input_file_pattern="${MSCOCO_DIR}/train-?????-of-00256"
   --inception_checkpoint_file="${INCEPTION_CHECKPOINT}"
   --train_dir="${MODEL_DIR}/train"
   --train_inception=false
   --number_of_steps=1000000

Step 5. Collect the running results

When the workload run is complete, it will display the output.

2) PyTorch based

Step 1. Required Software Stacks

torch
torchvision
matplotlib
$ pip install softwareName

**Step 2. Get workloads from BigDataBench**

Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0.AI_ComponentBenchmark.tar.gz

$ tar xf BigDataBench_V5.0.AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0.AI_ComponentBenchmark/
$ cd PyTorch/Image_to_Text

**Step 3. Prepare the input**

This can be done by running $ ./prepareData.sh or following the steps bellow:

1. Download from BigDataBench at the folder of ‘DataSet/coco2014’. Rename ‘coco2014’ as ‘data’ and then put ‘data’ in the folder of ‘Image_to_Text’.
2. Download from the website by running:
   $ ./download.sh

Preprocessing
$ python build_vocab.py
$ python resize.py

**Step 4. Run the workload**

This can be done by running $ ./run_imageToText.sh or following the steps bellow:

Train the model
$ python train.py

Test the model
$ python sample.py --image='png/example.png'

**Step 5. Collect the running results**

When training the model, it will display the output:

Epoch [0/1], Step [0/3236], Loss: 9.2094, Perplexity: 9990.5262
Epoch [0/1], Step [10/3236], Loss: 5.8074, Perplexity: 332.7434
...

At each ‘--save_step’, it will save the training model in the folder ‘./models’ with the name of ‘encoder-{epoch}-{step}.ckpt’ and ‘decoder-{epoch}-{step}.ckpt’. The default ‘--num_epochs’ is 5, ‘--save_step’ is 1000.

Test the model, the output is something like the following sentence.

<start> a man is sitting on a tennis court . <end>

The default training model used is ‘encoder-2-1000.ckpt’ and ‘decoder-2-1000.ckpt’, which can be changed by ‘--encoder_path’ and ‘--encoder_path’. 
4.3.5  Image to Image

1) TensorFlow based

Step 1. Required Software Stacks
- Python3
- Tensorflow1.2
- click (pip install click)
- unzip

Step 2. Get workloads from BigDataBench

Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz

```
$ tar –xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd TensorFlow/Image_to_Image/CycleGAN
```

Step 3. Prepare the input

```
$ ./download_datasets.sh cityscapes
```

Step 4. Run the workloads;

Add `export LC_ALL=C.UTF-8 export LANG=C.UTF-8` to `/etc/profile`

```
$./run.sh
```

Step 5. Collect the running results

When the workload run is complete, it will display the output.

2) PyTorch based

Step 1. Required Software Stacks
- PyTorch
- PyTorchvision
- PyDominate
- PyVisdom

Step 2. Get workloads from BigDataBench

Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz

```
$ tar –xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd PyTorch/Image_to_Image
```

Step 3. Prepare the input

The dataset we use is cityscapes.

You can download the dataset by:

```
$ bash ./datasets/download_cyclegan_dataset.sh cityscapes
```
Or you can also download the cityscapes dataset from http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet

Step 4. Run the workload

$ python train.py --dataroot ./datasets/cityscapes --name cityscapes_cycle_gan --model cycle_gan

Step 5. Collect the running results

When the workload run is complete, it will display the output.

4.3.6 Speech to Text

1) TensorFlow based

Step 1. Required Software Stacks

Python 2/3
Tensorflow>=1.1
$ pip/pip3 install -r requirements.txt

Step 2. Get workloads from BigDataBench

Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ tar --xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd TensorFlow/Speech_to_Text

Step 3. Prepare the input

$ python data/download.py
Arguments:
--data_dir: Directory where to download and save the preprocessed data. By default, it is /tmp/librispeech_data.
Use the --help or -h flag to get a full list of possible arguments.

Step 4. Run the workloads;

$python deep_speech.py
Arguments:
--model_dir: Directory to save model training checkpoints. By default, it is /tmp/deep_speech_model/.
--train_data_dir: Directory of the training dataset.
--eval_data_dir: Directory of the evaluation dataset.
--num_gpus: Number of GPUs to use (specify -1 if you want to use all available GPUs).

There are other arguments about DeepSpeech2 model and training/evaluation process. Use the --help or -h flag to get a full list of possible arguments with detailed descriptions.

Step 5. Collect the running results
A shell script run_deep_speech.sh is provided to run the whole pipeline with default parameters. Issue the following command to run the benchmark:
```
sh run_deep_speech.sh
```

Note by default, the training dataset in the benchmark include train-clean-100, train-clean-360 and train-other-500, and the evaluation dataset include dev-clean and dev-other.

When the workload run is complete, it will display the output.

2) **PyTorch based**

**Step 1. Required Software Stacks**

PyTorch 1.0.1, Torchaudio, apex, warp-ctc bindings, flac, sox, tqdm, librosa, levenshtein.

Cuda 10.0+

Since the torchaudio has high compatibility requirements, we suggest using conda to create a new environment and install the speech_to_text workload. Otherwise, “import torchaudio” would report segmentation fault (core dumped). The installation processes are as follows:

i) create a new environment named imageText, note that you can change the name.
```
$ conda create –n imageText python
```

ii) activate the new environment
```
$ source activate imageText
```

iii) install PyTorch in the new environment
```
$ conda install pytorch torchvision -c pytorch
```

iv) install torchaudio
```
$ git clone https://github.com/pytorch/audio.git
$ cd audio && python setup.py install
```

v) install apex
```
$ git clone --recursive https://github.com/NVIDIA/apex.git
$ cd apex && pip install .
```

vi) install warp-ctc binding
```
$ git clone https://github.com/SeanNaren/warp-ctc.git
$ cd warp-ctc
$ mkdir build
$ cd build
$ cmake ..
$ make
```

Note that the conda environment may install the gcc 6+ version, while warp-ctc doesn’t support gcc 6+, so you need to edit the CMakeLists.txt file to use old gcc version. Insert the following two lines in CMakeLists.txt (you need to change the path of old gcc version according to your environment) and then repeat the upper commands.

```
SET(CMAKE_C_COMPILER "/usr/bin/gcc4.8")
SET(CMAKE_CXX_COMPILER "/usr/bin/g++4.8")
```

vii) install pytorch_binding
```
$ cd warp-ctc/pytorch_binding
$ python setup.py install
```
viii) install flac
$ wget https://ftp.osuosl.org/pub/xiph/releases/flac/flac-1.2.1.tar.gz
$ tar --xf flac-1.2.1.tar.gz
$ cd flac-1.2.1
$ ./configure && make && make install
Note that if you encounter the error “main.cpp:75:27: error: 'memcmp' was not declared in this scope”, you need to insert “#include <string.h> ” in the file “examples/cpp/encode/file/main.cpp”.
ix) install sox
Download sox-14.4.2.tar.gz from https://sourceforge.net/projects/sox/files/sox/14.4.2/sox-14.4.2.tar.gz/download

$ ./configure --with-lame --with-flac --with-libvorbis
$ make -s
$ make install
x) install tqdm, librosa, and levenshtein
$ pip install tqdm
$ pip install librosa
$ pip install python-levenshtein

Step 2. Get workloads from BigDataBench
Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ tar --xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
Step 3. Prepare the input
The workload use LibriSpeech dataset. Preprocess the dataset:
$ cd deepspeech.pytorch/data
$ mkdir LibriSpeech_dataset
$ mkdir LibriSpeech_dataset/test_clean
$ mkdir LibriSpeech_dataset/test_other
$ mkdir LibriSpeech_dataset/train
$ mkdir LibriSpeech_dataset/val
$ python librispeech.py
you can use the parameter --files-to-use to specify the dataset.
The command will generate four files under the data directory:
libri_test_clean_manifest.csv,
libri_test_other_manifest.csv,
libri_train_manifest.csv,
libri_val_manifest.csv

Step 4. Run the workload
Training use CPU:
$ python train.py --train-manifest data/libri_train_manifest.csv --val-manifest data/libri_val_manifest.csv
Training use GPU:

```bash
$ python train.py --train-manifest data/libri_train_manifest.csv --val-manifest data/libri_val_manifest.csv --cuda
```

Testing use CPU:

```bash
$ python test.py --model-path models/deepspeech_final.pth --test-manifest data/libri_test_clean_manifest.csv
```

Testing use GPU:

```bash
$ python test.py --model-path models/deepspeech_final.pth --test-manifest data/libri_test_clean_manifest.csv --cuda
```

**Step 5. Collect the running results**

When the workload run is complete, it will display the output.

---

**4.3.7 Face Embedding**

**1) TensorFlow based**

**Step 1. Required Software Stacks**

- Tensorflow
- Scipy
- Scikit-learn
- Opencv-python
- H5py
- Matplotlib
- Pillow
- Requests
- Psutil

**Step 2. Get workloads from BigDataBench**

Download the Benchmark from the link:

http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz

```bash
$ tar -xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd TensorFlow/Face_embedding
```

**Step 3. Prepare the input**

The dataset we use is VGGFace2. http://zeus.robots.ox.ac.uk/vgg_face2/login/.

You can download dataset from VGGFace2 webpage, or you can also download the cityscapes dataset from:

http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet.

After downloading the dataset, you also need to perform the image alignment, which might take several hours.

```bash
$ ../scripts/face-align-VGGFace2.sh
```

**Step 4. Run the workloads**

```bash
$ ../scripts/cls_training_triplet_webface.sh
```

**Step 5. Collect the running results**
When the workload run is complete, it will display the output.

2) **PyTorch based**

**Step 1. Required Software Stacks**

Pytorch 1.0.1

**Step 2. Get workloads from BigDataBench**

Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz

```
$ tar –xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd PyTorch/Face_embedding/facenet
```

**Step 3. Prepare the input**

Rewrite `datasets/write_csv_for_making_dataset.py`, you need to change `which_dataset` and `root_dir`.

**Step 4. Run the workload**

```
$ python train.py
```

**Step 5. Collect the running results**

When the workload run is complete, it will display the output.

4.3.8 Object Detection

1) **TensorFlow based**

**Step 1. Required Software Stacks**

Python 3.3+;
OpenCV;
TensorFlow>1.6;

**Step 2. Get workloads from BigDataBench**

Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz

```
$ tar –xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd TensorFlow/Object_detection
```

**Step 3. Prepare the input**

To prepare the data:
```
./prepareData.sh
```

**Step 4. Run the workloads**;

i) To train on a single machine:
```
./run_objectDetect.sh
```
ii) To run distributed training:
Set TRAINER=horovod in the config.py file
./run_objectDetect.sh

**Step 5. Collect the running results**
When the workload run is complete, it will display the output.

**2) PyTorch based**

**Step 1. Required Software Stacks**
PyTorch
CyThon
Cffi
Opencv-python
Scipy
Msgpack
Easydict
Matplotlib
Pyyarml
TensorboardX

**Step 2. Get workloads from BigDataBench**
Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz

$ tar -xf BigDataBench_V5.0_AI_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_AI_ComponentBenchmark/
$ cd PyTorch/Object_detection

**Step 3. Prepare the input**
You can download the dataset via COCO webpage, or http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_DataSet.
After downloading the dataset, please execute the following command.

$ # set your own coco dataset path
$ COCO_PATH=/your/path/to/coco2014
$ mkdir -p data/data/pretrained_model
$ set -x
$ if [[ ! -d data/coco ]]; then
  $ cd data
git clone https://github.com/pdollar/coco.git && cd coco/PythonAPI
make -j32 && cd ../../
$ cd ../
$ fi
$ if [[ ! -f data/coco/annotations || ! -h data/coco/annotations ]]; then
  $ ln -sv $COCO_PATH/annotations data/coco/annotations
$ fi
Step 4. Run the workload

`./scripts/train.sh`

Step 5. Collect the running results

When the workload run is complete, it will display the output.

4.3.9 Recommendation - CF

1) Hadoop based

Step 1. Required Software Stacks

Java JDK

Step 2. Get workloads from BigDataBench

Download the Benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Step 3. Prepare the input

```
$ tar --xf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Hadoop/CF
$ ./genData-cf.sh <size>
```

The parameter “size” means the input data size (GB).

Step 4. Run the workloads;

```
$ ./run-cf.sh <size> <numFeatures> <numIterations> <lambda>
```

#size: the input data size, GB  
#numFeatures: the number of features  
#numIterations: the number of features  
#lambda: regularization parameter

Step 5. Collect the running results

When the workload run complete, it will display the running information and generate output file: /hadoop/cf/cf-out

2) Spark based

Step 1. Required Software Stacks

CentOS

Step 2. Get workloads from BigDataBench

Download the Benchmark from this link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Step 3. Prepare the input


```bash
$ tar -xf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Spark/CF
$ ./genData-cf.sh <size>
```

The parameter “size” means the input data size (GB).

**Step 4. Run the workload**

```bash
$ ./runSpark-cf.sh <ratings_file> <rank> <iterations>
```

parameters:

- `<ratings_file>`: path of input data file
- `<rank>`: number of features to train the model
- `<iterations>`: number of iterations to run the algorithm

**Step 5. Collect the running results**

When the workload run is complete, it will display the running information.

---

### 4.3.10 PageRank

1) **Hadoop based**

**Step 1. Required Software Stacks**

Hadoop
BGDS

**Step 2. Get workloads from BigDataBench**

Download the benchmark from the link:

http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Decompress the Hadoop package.

```bash
$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
```

Prepare:

```bash
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Hadoop
$ ./prepar.sh
```

**Step 3. Prepare the input**

```bash
$ cd PageRank/
$ ./genData-pagerank.sh <log_vertex>
```

Parameter “log_vertex” indicates that the vertex of the generated graph is $2^I$.

The generated data is on HDFS under /hadoop/pagerank directory.

**Step 4. Run the workload**

```bash
$ ./run-Pagerank.sh <log_vertex>
```

Parameter “log_vertex” indicates that the vertex of the generated graph is $2^I$.

**Step 5. Collect the running results**

The output of the workload will be put in hdfs with location:
/hadoop/pagerank/output.

2) **Spark based**

**Step 1. Required Software Stacks**
**Spark**

**BGDS**

**Step 2. Get workloads from BigDataBench**

Download the benchmark from the link:

Decompress the package.

$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

**Step 3. Prepare the input**

$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Spark

$ cd Pagerank

$ ./genData-pagerank.sh <log_vertex>

Parameter “log_vertex” indicates that the vertex of the generated graph is $2^I$.

The generated data is on HDFS under /spark/pagerank directory.

**Step 4. Run the workload**

$ ./runSpark-PageRank.sh <log_vertex>

Parameter “log_vertex” indicates that the vertex of the generated graph is $2^I$.

**Step 5. Collect the running results**

The output of the workload will be put in hdfs with location: /spark/pagerank/output.

---

**Flink based**

**Step 1. Required Software Stacks**

Flink

BGDS

**Step 2. Get workloads from BigDataBench**

Download the benchmark from the link:

Decompress the Hadoop package.

$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Prepare:

$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/

$ ./prepar.sh

**Step 3. Prepare the input**

$ cd Flink/Pagerank/

$./genData_PageRank.sh

You need to input the iteration $I$ of GenGraph, and it indicates that the vertex of the generated graph is $2^I$.

The generated data is on HDFS under /data-PageRank/Google_genGraph_$I.txt directory.
Step 4. Run the workload

$ ./run_Pagerank.sh
You need to input the iteration $I$ of GenGraph, and it indicates that the vertex of the generated graph is $2^I$.

Step 5. Collect the running results
The output of the workload will be put in hdfs with location: /flink-pagerank-result.

4) MPI based

MPI_Pagerank is a parallel implementation of pagerank algorithm.

Step 1. Required software stacks

MPICH2
Cmake: Cmake 2.8.12.2 is preferred 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_V5.0_BigData_ComponentBenchmark/MPI
$ cd Pagerank/parallel-bgl -0.7.0
$ cmake .
$ cd parallel-bgl-0.7.0/libs/graph_parallel/test
$ make distributed_page_rank_test

Step 2. Get workload MPI_Pagerank from BigDataBench
Download the benchmark from the link: http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Step 3. Prepare the input
The data set used by MPI_Pagerank is generated by big data generation tool (BDGS).
To generate data:
i) Unpack the downloaded tar file
$ tar –xf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/MPI/MPI_Pagerank
ii) Generate data
$ ./genData_PageRank.sh
Input the Iterations of GenGragh, after this step, it will generate data-PageRank under the Pagerank directory.

Step 4. Run the workload
i) Unpack the downloaded tar file
$ tar –xf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/MPI/MPI_Pagerank
ii) Install Pagerank
We provide a Compiled executable program named run_PageRank under the Pagerank directory.

iii) Run the workload

Run MPI Pagerank, command is:

```
$mpirun -f machine_file -n PROCESS_NUM ./run_PageRank InputGraph-file num_ofVertex num_ofEdges iterations
```

**Step 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.

**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 Pagerank]# ./genData_PageRank.sh
```

Generate PageRank data

Please Enter The Iterations of GenGragh: 5

```
WORK_DIR=/BigDataBench_V5.0_BigData_ComponentBenchmark/MPI/Pagerank
```

data will be generated under Pagerank/data-PageRank

```
sh: gnuplot: command not found
```


```
========================================
```

Output graph file name (-o):= 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)

4.3.11 Index

**1) Hadoop based**

**Step 1. Required Software Stacks**
Step 2. Get workloads from BigDataBench

Download the benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Decompress the package.
$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Prepare:
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark
$ ./prepar.sh

Step 3. Prepare the input

When prepare the input file linux.words and words, you should put them in directory /usr/share/dict.
$ cd Hadoop/Index/bin
$ ./genData_Index.sh

Then you will be asked how many data you like to generate:
Preparing MicroBenchmarks data dir
WORK_DIR=/BigDataBench_V5.0_BigData_ComponentBenchmark/Hadoop/Ind
ex data will be put in Index/data-Index print data size GB : (enter a number here)

Step 4. Run the workload

$ cd Hadoop/Index/bin
$ ./run_Index.sh

Step 5. Collect the running results

The output of the workload will be put in local directory: result and the output is redirected to file: Index.out

4.3.12 BFS

1) MPI based

MPI_BFS is an MPI-based implementation of breadth-first search.

Step 1. Required software stacks

MPICH2

Step 2. Get workload MPI_BFS from BigDataBench

Download the benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Step 3. Prepare the input

The data set used by BFS is generated by the program itself.

Step 4. Run the workload

i) Unpack the downloaded tar file
$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/MPI/
$ cd BFS-MPI/graph500

ii) 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 directory.

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/graph500/mpi.

iii) 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 2SCALE. 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 2SCALE). 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

Step 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
consstruction_time: 54.5597 s
min_time: 0.287835 s
......
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.

4.3.13 K-means

1) Hadoop based
Step 1. Required Software Stacks
Hadoop
BGDS

Step 2. Get workloads from BigDataBench
Download the benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
Decompress the Hadoop package.
$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Hadoop

Step 3. Prepare the input
$ cd Kmeans/
$./genData-kmeans.sh <log_vertex>
Parameter “log_vertex” indicates that the vertex of the generated graph is $2^I$.
The generated data is on HDFS under /user/root/testdata directory.

Step 4. Run the workload
$ ./run-Kmeans.sh <t1> <t2> <cd> <x>
Parameter:
#   t1: T1 threshold value (0-1), such as 0.4
#   t2: T2 threshold value (0-1), such as 0.1
#   cd: The convergence delta value (0-1), such as 0.1
#   x: The max iteration number

Step 5. Collect the running results
The output of the workload will be put in hdfs with location: /user/root/output.

2) Spark based

Step 1. Required Software Stacks
Spark
BGDS

Step 2. Get workloads from BigDataBench
Download the benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
Decompress the Hadoop package.
$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Spark

Step 3. Prepare the input
$ cd Kmeans/
$./genData-kmeans.sh <size>
Parameter “size” indicates the input data size (GB).
The generated data is on HDFS under /spark/kmeans/ directory.
Step 4. Run the workload

$ ./runSpark-Kmeans.sh <size> <centerNum> <iterNum>

Parameter:
# size: the input data size (GB)
# centerNum: the number of center points.
# iterNum: the max iteration number

Step 5. Collect the running results
The output of the workload will be put in hdfs.

3) Flink based

Step 1. Required Software Stacks
Flink
BGDS

Step 2. Get workloads from BigDataBench
Download the benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
Decompress the package.
$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Flink

Step 3. Prepare the input
$ cd Kmeans/
$./genData_Kmeans.sh
Then you need to input the data size (GB) you want to generate. The data will be put under /Flink-Kmeans directory on HDFS.

Step 4. Run the workload
$ ./run_Kmeans.sh
You will need to input the number of centers and the number of max iterations.

Step 5. Collect the running results
The output of the workload will be put in hdfs.

4) MPI based

Step 1. Required software stacks
MPICH2

Step 2. Get workload MPI_Kmeans from BigDataBench
Download the benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Step 3. Prepare the input
The data set used by MPI_Kmeans is generated by a generating script.
To generate data:
i) Unpack the downloaded tar file
   $ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
   $ cd BigDataBench_V5.0_BigData_ComponentBenchmark/
   $ cd MPI/Simple_Kmeans
   ii) 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 generate 10 GB data. After this step, it will generate data-Kmeans file under directory of Simple_Kmeans.

**Step 4. Run the workload**

   $cd MPI/Simple_Kmeans

Install MPI_Kmeans
   $make
   After this step, there will be an executable files named mpi_main under the current directory.

Run MPI_Kmeans command:
   $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 written to file "data-Kmeans.cluster_centres", and the membership of all data objects are written 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) ****
   ...... 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.

**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

4.3.14 NaiveBayes
The Naive Bayes is a simple probabilistic classifier, which applies the Bayes’ theorem with strong (naive) independency assumptions.

1) Hadoop based

Step 1. Required Software Stacks
Hadoop
BDGS

Step 2. Get workloads from BigDataBench
Download the benchmark from the link:
http://prof.ict.ac.cn/bdb/uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
Decompress the Hadoop package.

$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Step 3. Prepare the input

$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Hadoop/Bayes
$ ./genData-bayes.sh <size>
The parameter “size” means the input data size (GB).
The data will be generated in /hadoop/Bayes/ on HDFS.

Step 4. Run the workload
Use the apache-mahout-0.10.2-compile under the Hadoop directory and set $MAHOUT_HOME in ~/.bashrc file.

$ ./run-bayes.sh

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

2) Spark based

Step 1. Required Software Stacks
Spark
BDGS

Step 2. Get workloads from BigDataBench
Download the benchmark from the link:
http://prof.ict.ac.cn/bdb/uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Step 3. Prepare the input

$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Spark/Bayes
$ ./genData-bayes.sh <size>
The parameter “size” means the input data size (GB).
The data will be generated in /spark/Bayes/ on HDFS.

Step 4. Run the workload

$ ./runSpark-bayes.sh <size>
The parameter “size” means the input data size (GB).
Step 5. Collect the running results
The output of the workload will be put in hdfs with location: /spark/Bayes/output.

3) Flink based

Step 1. Required Software Stacks
Flink
BDGS

Step 2. Get workloads from BigDataBench
Download the benchmark from the link: 
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigDataComponentBenchmark.tar.gz
Decompress the Hadoop package.
$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Step 3. Prepare the input
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Flink/naivebayes
$ ./genData_naivebayes.sh
You need to input the data size (GB) you want to generate. The data will be generated in /Bayesclassifier/testdata on HDFS.

Step 4. Run the workload
$ ./run_naivebayes.sh

Step 5. Collect the running results
The output will be put on /flink-Bayes-result directory.

4) MPI based
MPI_NaiveBayes is a mpi-based implementation of naive bayes algorithm.

Step 1. Required software stacks
MPICH2

Step 2. Get workload MPI_NaiveBayes from BigDataBench
Download the benchmark from the link:  
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigDataComponentBenchmark.tar.gz

Step 3. Prepare the input
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/MPI/MPI_naivebayes
$ ./genData_naivebayes.sh
Then you will be asked how many data you would like to generate: Preparing naivebayes-naivebayes data dir WORK_DIR=Naviebayes will be generated in Naviebayes/data-naivebayes
Preparing naivebayes-naivebayes data dir print data size GB : (enter a number here)

Step 4. Run the workload
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.

4.3.15 LDA
1) Spark based

Step 1. Required Software Stacks
Spark
BDGS

Step 2. Get workloads from BigDataBench
Download the benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
Decompress the Hadoop package.
$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Step 3. Prepare the input
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Spark/LDA
$ ./genData-lda.sh <size>
The parameter “size” means the input data size (GB).
The data will be generated in /spark/lda/wiki"G" on HDFS.
After the generation, dictionary file and corpus file will be generated on HDFS:
/spark/lda/dictionary /spark/lda/corpus

Step 4. Run the workload
$ ./runSpark-lda.sh

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

2) MPI based

Step 1. Required Software Stacks
MPICH2
BDGS
**Step 2. Get workloads from BigDataBench**

Download the benchmark from the link: [http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz](http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz)

Decompress the Hadoop package.

```
$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
```

**Step 3. Prepare the input**

```
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/BigDataBench_V5.0_BigData_ComponentBenchmark
$ ./genData.sh <size>
```

The parameter “size” means the input data size (GB).

**Step 4. Run the workload**

```
$ ./run-mpiLDA.sh <size>
```

The parameter “size” means the input data size (GB), which is the same with the generating command.

**Step 5. Collect the running results**

The output will be printed on the screen.

---

### 4.3.16 SIFT

1) **Hadoop based**

**Step 1. Required Software Stacks**

Hadoop

**Step 2. Get workloads from BigDataBench**

Download the benchmark from the link: [http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz](http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz)

Decompress the Hadoop package.

```
$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
```

**Step 3. Prepare the input**

```
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/Hadoop/SIFT
$ ./genData.sh <size>
```

SIFT workload uses ImageNet dataset, and the dataset is put under Hadoop/SIFT/hadoop-SIFT/data directory.

**Step 4. Run the workload**

```
$ ./run-sift.sh <imgsize>
```

The parameter “imgsize” indicates the input image size (GB).

**Step 5. Collect the running results**

The output will be printed on the screen.

2) **MPI based**
MPI SIFT workload is an adaptation of David Lowe's source code, which detects and describes local features in input images. We modified it to a data parallel version using MPI.

**Step 1. Required software stacks**

MPICH2
OpenCV package http://sourceforge.net/projects/opencvlibrary/
GDK/GTK+2 http://www.gtk.org/

**Step 2. Get workload SIFT from BigDataBench**

Download the benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

**Step 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.

**Step 4. Run the workload**

Unpack the downloaded tar file

$ tar -zxvf BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz
$ cd BigDataBench_V5.0_BigData_ComponentBenchmark/MPI/mpiSIFT

Build the MPI executables

$ 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

**Step 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.

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.

4.3.17 DBN

1) MPI based

DBN workload is a MPI implementation of deep belief networks.

Step 1. Required software stacks

MPICH2

Step 2. Get workload DBN from BigDataBench

Download the benchmark from the link:
http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/BigDataBench_V5.0_BigData_ComponentBenchmark.tar.gz

Step 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.

Step 4. Run the workload

Unpack the downloaded tar file
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, Stacked RBMS 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:
   $ 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.

Step 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.

For more information about DBN workload, please refer to: MPI/DBN/README

4.4 Application Benchmarks

4.4.1 DCMix

4.4.1.1 Introduction
Modern datacenter computer systems are widely deployed with mixed workloads to improve system utilization and save cost. However, the throughput of latency-critical workloads is dominated by their worst-case performance-tail latency. To model this important application scenario, we propose an end-to-end application benchmark---DCMix to generate mixed workloads whose latencies range from microseconds to minutes with four mixed execution modes.

4.4.1.2 DCMix Framework
There are four main modules: Workloads, User interface, mixed workloads generator, and Performance monitor. DCMIX contains two types of workloads: online service and data analytic workloads and they are all deployed on the target system. User interface is the portal for user; users can specify their workload mix requirements, including workloads and mixture patterns. Mixed workloads generator can generate the mixed workloads through submitting queries (service requests queries and data analytics job submitting queries). Performance monitor can monitor the performance data of the target system, and the system entropy is calculated by these original monitor data.

1) Workload Overview

DCMIX contains two types of workloads: online service and data analytic workloads. These workloads have different application fields and different user experience (latency). DCMIX’s application fields are big data, artificial intelligence, high-performance computing, transaction processing databases, et al. The latencies of DCMIX workloads range from microseconds to minutes.

2) Mixed Workload Generator
Mixed workloads generator can generate the mixed workloads through submitting queries (service requests queries and data analytics job submitting queries). Mixed workloads generator supports the mixture execution of serial execution and parallel execution. Serial execution means that the workload must start up after the previous workload complete. Parallel execution means that multiple workloads start up at the same time. Moreover, in the workload generator configuration file, users can set request configurations for each workload. For online-services, we provided request intensity, number of requests, number of warm-up requests, etc.; for offline-analytics, we provide path of the data set, threads number of jobs, etc.

4.4.1.3 How to use
Step 1. Required software
Python, gcc, gcc-c++, make, automake, autoconf, epel-release, libtool, libuuid, e2fsprogs, openssl, bison, swig, boost-devel, readline-devel, libdb-cxx-devel, numactl-devel, libaio-devel

Step 2. Download DCMix
You can download DCMIX via http://prof.ict.ac.cn/bdb_uploads/bdb_5/packages/DCMIX.tar.gz

Step 3. Install DCMix
1. To install tailbench workloads (i.e, online services):
   $ cd tailbench-v0.9
   $ bash ./build.sh
2. To install dwarf workloads (i.e, offline applications):
   $ cd dwarf-set
   $ bash ./build.sh

Step 4. Prepare the input data
1. To get tailbench dataset:
   $ mkdir -p tailbench-data
   $ wget -c http://tailbench.csail.mit.edu/tailbench.inputs.tgz
2. To generate dwarf dataset:
   $ g++ -std=c++11 gen-data.cpp -o gen-data
   $ ./gen-data

Step 5. Run DCMix Workload
1. To run tailbench workloads, let’s take xapian as an example:
   $ cd tailbenchv-0.9/xapian
   $ ./run_xapian_server.sh
   $ ./run_xapian_client.sh
   You can set the request parameters in the above 2 script files.
2. To run dwarf workloads:
   $ ./run_all.sh

4.5 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.

4.5.1 Environment setup

Step 1. Versions of software
CentOS 6.0
JKD 1.7
Python 2.7

Step 2. Hadoop cluster setup
Refer to http://hadoop.apache.org/#Getting+Started

Step 3. Environment setup of Nutch search engine
Refer to http://prof.ict.ac.cn/DCBenchmarks/Search_manual_v1.0.pdf
Search source code download: http://prof.ict.ac.cn/DCBenchmarks
Note: In Search installation, if using normal user to login, you need to set password-free logins

Step 4. Shark environment setup
Referred to https://github.com/amplab/shark/wiki/Running-Shark-on-a-Cluster

Step 5. Environment variable configuration
Configure variables at /etc/profile

```
HADOOP_HOME=/opt/hadoop-1.2.1
SEARCH_HOMES=/opt/search/search
```

Step 6. Copy the configuration file to $HADOOP_HOME/conf

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

4.5.2 Installation and Configuration of Software

Step 1: Download and unload the package of software
mixWorkloadSuite.tar at tmp form

Step 2: 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} -d hdfsWrite WriteToHdfs.java jar -cvf WriteToHdfs.jar -C hdfsWrite/.
```

Step 3: 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 4: Execute the following commands**

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

### 4.5.3 Generate the replay script

**Step 1. Obtain a representative load**

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

```bash
$ 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 tab 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:**

```bash
$ 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.

Import data [Workload trace processing] to give the log file [cleanup workload trace, extract the information needed]:

```bash
$ FB-2009_samplesBySort_24_times_1hr_0.tsv outputFILE.tsv
```

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

```bash
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:**

```bash
$ python k_means_FB.py logFile.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:

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

**Example of use:**

```bash
$ ./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.

Example of use:

   $python get_optimal_K.py /opt/mixWorkloadSuite/logfile/

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 2: Use GenerateReplayScriptFB.java to create a folder that in-cludes 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: [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 3: 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 command]
```

4.5.4 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 argument(f/g or m)
```

4.6 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
$ 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.

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.

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

3) Workloads running


$ 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

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

<table>
<thead>
<tr>
<th>Workload</th>
<th>Master</th>
<th>Slaver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hive-Differ</td>
<td><code>cd /master</code></td>
<td><code>cd /slaver</code></td>
</tr>
<tr>
<td></td>
<td>./simics HiveDiffer_L</td>
<td>./simics HiveDiffer_LL</td>
</tr>
<tr>
<td></td>
<td>./BigOP-e-commerce-difference.sh</td>
<td></td>
</tr>
<tr>
<td>Hive-TPC-DS-query3</td>
<td><code>cd /master</code></td>
<td><code>cd /slaver</code></td>
</tr>
<tr>
<td></td>
<td>./simics -c</td>
<td>./simics -c</td>
</tr>
<tr>
<td></td>
<td>Hadoopgrep_L</td>
<td>Hadoopgrep_LL</td>
</tr>
<tr>
<td></td>
<td>./query3.sh</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Workload</th>
<th>Master</th>
<th>Slaver</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark-WordCount</td>
<td><code>cd /master</code></td>
<td><code>cd /slaver</code></td>
</tr>
<tr>
<td></td>
<td>./simics -c</td>
<td>./simics -c</td>
</tr>
<tr>
<td></td>
<td>SparkWordcount_L</td>
<td>SparkWordcount_LL</td>
</tr>
<tr>
<td></td>
<td>./run-bigdatabench</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cn.ac.ict.bigdatabench.WordCount</td>
<td></td>
</tr>
<tr>
<td></td>
<td>spark://10.10.0.13:7077</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/in /tmp/wordcount</td>
<td></td>
</tr>
<tr>
<td>Spark-Grep</td>
<td><code>cd /master</code></td>
<td><code>cd /slaver</code></td>
</tr>
<tr>
<td></td>
<td>./simics -c</td>
<td>./simics -c</td>
</tr>
<tr>
<td></td>
<td>Sparkgrep_L</td>
<td>Sparkgrep_LL</td>
</tr>
<tr>
<td>Workload</td>
<td>Master</td>
<td>Slaver</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>Spark-Sort</td>
<td>cd /master</td>
<td>cd /slaver</td>
</tr>
<tr>
<td></td>
<td>./simics -c SparkSort_L</td>
<td>./simics -c SparkSort_LL</td>
</tr>
<tr>
<td>Spark-Pagerank</td>
<td>cd /master</td>
<td>cd /slaver</td>
</tr>
<tr>
<td></td>
<td>./simics -c SparkPagerank_L</td>
<td>./simics -c SparkPagerank_LL</td>
</tr>
<tr>
<td>Spark-Kmeans</td>
<td>cd /master</td>
<td>cd /slaver</td>
</tr>
<tr>
<td></td>
<td>./simics -c SparkKmeans_L</td>
<td>./simics -c SparkKmeans_LL</td>
</tr>
<tr>
<td></td>
<td>/run-bigdatabench cn.ac.ict.bigdatabench.Grep</td>
<td>/run-bigdatabench cn.ac.ict.bigdatabench.Sort</td>
</tr>
<tr>
<td></td>
<td>/run-bigdatabench cn.ac.ict.bigdatabench.Sort</td>
<td>/run-bigdatabench cn.ac.ict.bigdatabench.Pagerank</td>
</tr>
<tr>
<td></td>
<td>spark://10.10.0.13:7077 /Google_genGraph_5.txt /tmp/PageRank</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/run-bigdatabench org.apache.spark.mllib.clustering.KMeans</td>
<td></td>
</tr>
<tr>
<td></td>
<td>spark://10.10.0.13:7077 /data 8 4</td>
<td></td>
</tr>
</tbody>
</table>

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 |
<table>
<thead>
<tr>
<th></th>
<th>./simics -c Sharkprojectorder_L</th>
<th>./runMicroBenchmark mark.sh</th>
<th>./simics -c Sharkprojectorder_LL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shark-TPC-DS-query8</td>
<td>cd /master</td>
<td></td>
<td>cd /slaver</td>
</tr>
<tr>
<td></td>
<td>./simics -c Sharkproquery8_L</td>
<td></td>
<td>./simics -c Sharkquery8_LL</td>
</tr>
<tr>
<td></td>
<td>shark -f query8.sql</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shark-TPC-DS-query10</td>
<td>cd /master</td>
<td></td>
<td>cd /slaver</td>
</tr>
<tr>
<td></td>
<td>./simics -c Sharkproquery10_L</td>
<td></td>
<td>./simics -c Sharkquery10_LL</td>
</tr>
<tr>
<td></td>
<td>shark -f query10.sql</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.7 Nutch Search Engine

4.7.1 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 4.7.2, for people who have never used Search before.

– How to install Search version 1.0 is introduced in 4.7.3, 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 4.7.4, 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/

4.7.2 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.

4.7.3 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 4.7.4.

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:
- Intel Xeon processor 3000, 3200, 5100, 5300 series
- Intel Core 2 duo processor
If you use other CPUs, you may go over the CPU part in 4.7.4.

**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 required:

```
$ 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
```
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 num-ber
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 data, 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@?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 fin-ished. 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.

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 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.

```bash
$ 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
- `corrcoeef( 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 "path" 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:

```python
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, $metric,$metric) 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

### 4.7.4 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 4.7.3, you should modify line 167 to line 201 of file $Search/hmon/hmon.py.

```python
kperf_events_map = " 
CPU_CLK_UNHALTED.CORE 3c # cpu_cycles
CPU_CLK_UNHALTED.BUS 13c # bus cycles
INST_RETIRED.ANY c0 # insets
ITLB_MISS_RETIRED c9 # itlb_misses
```
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:
1 server-list=gd87 gd88 gd89 gd90
2 gd87-crawl-dir=01
3 gd88-crawl-dir=23
4 gd89-crawl-dir=45
5 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 snap-shot and index.

### iii. Creating your own workload

4.7.3 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:

**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 parameter. 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 4.7.4.

4.7.5 Appendix A - Metrics collected by DCAngel
<table>
<thead>
<tr>
<th>variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpu_cycles</td>
<td>Core cycles when core is not halted</td>
</tr>
<tr>
<td>bus_cycles</td>
<td>Bus cycles when core is not halted</td>
</tr>
<tr>
<td>insts</td>
<td>Retired instructions</td>
</tr>
<tr>
<td>itlb_misses</td>
<td>Retired instructions that missed the ITLB</td>
</tr>
<tr>
<td>dtlb_misses</td>
<td>Memory accesses that missed the DTLB</td>
</tr>
<tr>
<td>icache_misses</td>
<td>Instruction Fetch Unit misses</td>
</tr>
<tr>
<td>dcache_misses</td>
<td>L1 data cache misses</td>
</tr>
<tr>
<td>page_walks</td>
<td>Duration of page-walks in core cycles</td>
</tr>
<tr>
<td>icache_stalls</td>
<td>Cycles during which instruction fetches stalled</td>
</tr>
<tr>
<td>br_insts</td>
<td>Retired branch instructions</td>
</tr>
<tr>
<td>br_misses</td>
<td>Retired mispredicted branch instructions.</td>
</tr>
<tr>
<td>load_insts</td>
<td>Instructions retired, which contain a load</td>
</tr>
<tr>
<td>store_insts</td>
<td>Instructions retired, which contain a store</td>
</tr>
<tr>
<td>other_insts</td>
<td>Instructions retired, which no load or store operation</td>
</tr>
<tr>
<td>simd_insts</td>
<td>Retired Streaming SIMD instructions</td>
</tr>
<tr>
<td>fp_insts</td>
<td>Floating point computational micro-ops executed</td>
</tr>
<tr>
<td>res_stalls</td>
<td>Resource related stalls</td>
</tr>
<tr>
<td>rob_stalls</td>
<td>Cycles during which the reorder buffer full</td>
</tr>
<tr>
<td>rs_stalls</td>
<td>Cycles during which the reserve station full</td>
</tr>
<tr>
<td>ldst_stalls</td>
<td>Cycles during which the pipeline has exceeded load or store limit or waiting to commit all stores</td>
</tr>
<tr>
<td>fpcw_stalls</td>
<td>Cycles stalled due to floating-point unit control word writes</td>
</tr>
<tr>
<td>br_miss_stalls</td>
<td>Cycles stalled due to branch misprediction</td>
</tr>
<tr>
<td>bus_trans</td>
<td>All bus transactions</td>
</tr>
<tr>
<td>bus_drdy</td>
<td>Bus cycles when data is sent on the bus</td>
</tr>
<tr>
<td>bus_bnr</td>
<td>Number of Bus Not Ready signals asserted</td>
</tr>
<tr>
<td>bus_trans_brd</td>
<td>Burst read bus transactions</td>
</tr>
<tr>
<td>bus_trans_rfo</td>
<td>Read For Ownership bus transactions</td>
</tr>
<tr>
<td>usr</td>
<td>User mode CPU time</td>
</tr>
<tr>
<td>nice</td>
<td>The CPU time of processes whose nice value is</td>
</tr>
<tr>
<td>sys</td>
<td>Kernel mode CPU time</td>
</tr>
<tr>
<td>idle</td>
<td>Idle time</td>
</tr>
<tr>
<td>iowait</td>
<td>Iowait time</td>
</tr>
<tr>
<td>irq</td>
<td>Hard interrupt time</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>softirq</td>
<td>Soft interrupt time</td>
</tr>
<tr>
<td>intr</td>
<td>The times of interrupt happened</td>
</tr>
<tr>
<td>ctx</td>
<td>Context switch times</td>
</tr>
<tr>
<td>proc</td>
<td>Process number</td>
</tr>
<tr>
<td>running</td>
<td>The number of processes that is running</td>
</tr>
<tr>
<td>blocked</td>
<td>The number of processes that is blocked</td>
</tr>
<tr>
<td>mem_total</td>
<td>Total memory</td>
</tr>
<tr>
<td>free</td>
<td>Memory that is not used</td>
</tr>
<tr>
<td>buffers</td>
<td>Size memory in buffer cache</td>
</tr>
<tr>
<td>cached</td>
<td>Memory that cache used</td>
</tr>
<tr>
<td>swap_cached</td>
<td>Memory that once was swapped out, but still in the swapfile</td>
</tr>
<tr>
<td>active</td>
<td>Memory that has been used more recently</td>
</tr>
<tr>
<td>inactive</td>
<td>Memory that is not active</td>
</tr>
<tr>
<td>swap_total</td>
<td>Total amount of physical swap memory</td>
</tr>
<tr>
<td>swap_free</td>
<td>Total amount of free swap memory</td>
</tr>
<tr>
<td>pgin</td>
<td>The number of pages that paged in from disk</td>
</tr>
<tr>
<td>pgout</td>
<td>The number of pages that paged out to disk</td>
</tr>
<tr>
<td>pgfault</td>
<td>The number of page fault</td>
</tr>
<tr>
<td>pgmajfault</td>
<td>The number of major page faults</td>
</tr>
<tr>
<td>active_conn</td>
<td>TCP active connection</td>
</tr>
<tr>
<td>passive_conn</td>
<td>TCP passive connection</td>
</tr>
<tr>
<td>rbytes</td>
<td>Received bytes</td>
</tr>
<tr>
<td>rpackets</td>
<td>Received packets</td>
</tr>
<tr>
<td>rerrs</td>
<td>Received error packets number</td>
</tr>
<tr>
<td>rdrop</td>
<td>Number of packets dropped by native network adapter</td>
</tr>
<tr>
<td>sbytes</td>
<td>Bytes sent</td>
</tr>
<tr>
<td>spackets</td>
<td>Packets sent</td>
</tr>
<tr>
<td>serrs</td>
<td>Number of error packets sent</td>
</tr>
<tr>
<td>sdrop</td>
<td>Number of packets dropped by remote network adapter</td>
</tr>
<tr>
<td>read</td>
<td>Times of disk reads</td>
</tr>
<tr>
<td>read_merged</td>
<td>Times of disk merged reads</td>
</tr>
<tr>
<td>read_sectors</td>
<td>Times of sectors read</td>
</tr>
<tr>
<td>read_time</td>
<td>The total time disk read</td>
</tr>
<tr>
<td>write</td>
<td>Times of disk writes</td>
</tr>
<tr>
<td>write_merged</td>
<td>Times of merged disk writes</td>
</tr>
<tr>
<td>write_sectors</td>
<td>Times of sectors write</td>
</tr>
<tr>
<td>write_time</td>
<td>The total time of disk write</td>
</tr>
</tbody>
</table>

### 4.7.6 Appendix B - DCAngel database table structure

For the meaning of all following table’s abbreviations, users can go over Appendix A.
### Table exps

<table>
<thead>
<tr>
<th>field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The test performance data’s path under exp/ directory</td>
</tr>
<tr>
<td>app</td>
<td>User used application’s name</td>
</tr>
<tr>
<td>comment</td>
<td>The comment when user used to specify a</td>
</tr>
<tr>
<td>reqs</td>
<td>Request name</td>
</tr>
<tr>
<td>duration</td>
<td>The test’s duration</td>
</tr>
<tr>
<td>host</td>
<td>Node’s host name</td>
</tr>
</tbody>
</table>

### Table _all

<table>
<thead>
<tr>
<th>Field</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The test performance data’s path under exp/ directory</td>
</tr>
<tr>
<td>host</td>
<td>Node’s host name</td>
</tr>
<tr>
<td>insts</td>
<td>The mean value of instruction number</td>
</tr>
<tr>
<td>cpi</td>
<td>Cycles per instruction</td>
</tr>
<tr>
<td>br_miss_ratio</td>
<td>Branch miss ratio</td>
</tr>
<tr>
<td>br_stall_ratio</td>
<td>Branch stall ratio</td>
</tr>
<tr>
<td>icache_stall_ratio</td>
<td>Icache stall ratio</td>
</tr>
<tr>
<td>tlb_stall_ratio</td>
<td>TLB stall ratio</td>
</tr>
<tr>
<td>dcache_stall_ratio</td>
<td>Dcache stall ratio</td>
</tr>
<tr>
<td>l2cache_stall_ratio</td>
<td>L2 Cache stall ratio</td>
</tr>
<tr>
<td>res_stall_ratio</td>
<td>Resource related stall ratio</td>
</tr>
<tr>
<td>rob_stall_ratio</td>
<td>Reorder buffer stall ratio</td>
</tr>
<tr>
<td>rs_stall_ratio</td>
<td>Reserve station stall ratio</td>
</tr>
<tr>
<td>ldst_stall_ratio</td>
<td>Load and store stall ratio</td>
</tr>
<tr>
<td>fpcw_stall_ratio</td>
<td>Float point unit stall ratio</td>
</tr>
<tr>
<td>br_mix</td>
<td>Branch instruction ratio</td>
</tr>
<tr>
<td>load_mix</td>
<td>Load instruction ratio</td>
</tr>
<tr>
<td>store_mix</td>
<td>Store instruction ratio</td>
</tr>
<tr>
<td>ldst_mix</td>
<td>Load and store instruction ratio</td>
</tr>
<tr>
<td>simd_mix</td>
<td>SIMD instruction ratio</td>
</tr>
<tr>
<td>fp_mix</td>
<td>Float point instruction ratio</td>
</tr>
<tr>
<td>other_mix</td>
<td>Instructions that except load and store ratio</td>
</tr>
<tr>
<td>bus_util</td>
<td>Bus utilization</td>
</tr>
<tr>
<td>bus_d_util</td>
<td>Bus_drdy ratio users can find bus_drdy and all the following abbreviations’ meaning in Appendix A</td>
</tr>
<tr>
<td>bus_bnr_ratio</td>
<td>bus_bnr ratio</td>
</tr>
<tr>
<td>bus_brdratio</td>
<td>bus_brdratio</td>
</tr>
<tr>
<td>bus_rfo_ratio</td>
<td>bus_rfo_ratio</td>
</tr>
<tr>
<td>cpu_usage</td>
<td>CPU utilization</td>
</tr>
<tr>
<td>search_latency</td>
<td>Average query latency</td>
</tr>
<tr>
<td>search_start</td>
<td>Test start time</td>
</tr>
<tr>
<td>duration</td>
<td>The test’s duration</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>netbytes</td>
<td>rnetbytes+snetbytes</td>
</tr>
<tr>
<td>netpackets</td>
<td>rnetpacket+snetpacket</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>macros</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$prim</td>
<td>app, comment, reqs, host</td>
</tr>
<tr>
<td>$hpc_basic</td>
<td>insts, cpi, br_miss_ratio</td>
</tr>
<tr>
<td>$stall_breakdown</td>
<td>br_stall_ratio, icache_stall_ratio, tlb_stall_ratio, l2cache_stall_ratio, dcache_stall_ratio, _ratio, res_stall rob_stall_ratio, _ratio, rs_stall_ratio, ldst_stall_ratio, fpcw_stall_ratio</td>
</tr>
<tr>
<td>$inst_mix</td>
<td>br_mix, load_mix, store_mix, ldst_mix, simd_mix, fp_mix, other_mix</td>
</tr>
<tr>
<td>$cache</td>
<td>itlb_miss ratio, dtlb_miss_ratio, icache_miss_ratio, dcache_miss _ratio, l2cache_miss_ratio</td>
</tr>
<tr>
<td>$bus</td>
<td>bus_util, bus_d_u, bus_bnr_ratio, bus_brd_ratio, bus_rfo_ratio</td>
</tr>
<tr>
<td>$proc_basic</td>
<td>cpu_usage, iowait, ctx, active, pgfault, pgmajfault</td>
</tr>
<tr>
<td>$net</td>
<td>active_conn, passive_conn, netbytes, netpackets,</td>
</tr>
<tr>
<td>$disk</td>
<td>read, write, read_sectors, write_sectors</td>
</tr>
<tr>
<td>$proc_selected</td>
<td>cpu_usage, iowait, ctx, active, pgmajfault, read_sectors</td>
</tr>
<tr>
<td>$hpc_all</td>
<td>$hpc_basic, $cache, $bus, $inst_mix</td>
</tr>
<tr>
<td>$proc_all</td>
<td>$proc_basic, $net, $disk</td>
</tr>
</tbody>
</table>