Big Data Bench 3.0

User's Manual

Contents

This document presents information on BigDataBench—— a big data benchmark suite from internet services, including a brief introduction and the usage of it. The information and specifications contained are for researchers who are interested in big data benchmarking.

Publishing information:
Release 3.0
Date 22/4/2014
Contact information:
Website: http://prof.ict.ac.cn/BigDataBench/

1 Background

As a multi-discipline---e.g., system, architecture, and data management---research effort, BigDataBench is a big data benchmark suite (please refer to our summary paper and presentation at HPCA 2014). It includes 6 real-world and 2 synthetic data sets, and 32 big data workloads, covering micro and application benchmarks from areas of search engine, social networks, e-commerce. In generating representative and variety of big data workloads, BigDataBench focuses on units of computation frequently appearing in Cloud “OLTP”, OLAP, interactive and offline analytics (Please refer to our DASFAA paper). BigDataBench also provides several (parallel) big data generation tools–BDGS– to generate scalable big data, e.g. PB scale, from small-scale real-world data while preserving their original characteristics. For example, on an 8-node cluster system, BDGS generates 10 TB data in 5 hours. For the same workloads, different implementations are provided. Currently, we and other developers implemented the offline analytics workloads using MapReduce, MPI, Spark, DataMPI, interactive analytics and OLAP workloads using Shark, Impala, and Hive.

2 Release and Update History

========2014.4.22 Version 3.0 Released, Current ========
Fix Bugs
18 OLAP and Interactive Query workloads
Parallel data generator tools

========2014.1.20 Version 2.2 Released, Current ========
Fix Bugs

========2013.11.22 Version 2.1 Released ========
Fix Bugs
Add Big Data Generate Suite

========2013.10.7 Version 2.0 Released ========

A big data benchmark suite from internet services.
New Big Data Generate Tools, and 19 Big Data Workloads with 6 raw data sets

========2013.5.30 Version 1.0 Released========
A big data benchmark suite from web search engines.

3 Introduction

3.1 Our Benchmarking Methodology

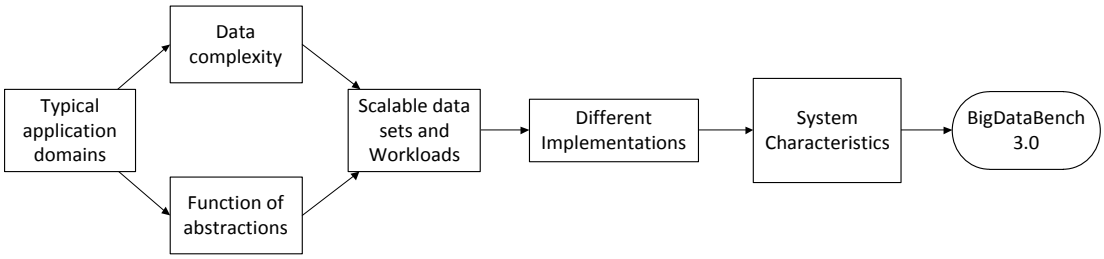

Figure 1 big data benchmarking methodology

Figure 1 summarizes the methodology of BigDataBench 3.0. Overall, it includes six steps: investigating typical application domains; understanding and constructing (or choosing) workloads via function of abstractions and data sets; generating scalable data sets and workloads; providing different implementations; system characterization; and finalizing benchmarks.
At the first step, we consider typical real-world data sets and representative big data analytics workloads which include typical internet service scenes such as Search engines, Social network and E-commercial. At the second step, we observe and identify a full spectrum of units of computation that frequently appear in big data analytics. Such as Projection Filter Aggregation Order by Cross Product Union Difference are the basic units of computation in the interactive workloads, and other interactive workloads can be presented by different units of computation. At the same time, we understand data sets from the perspectives of data natures, data sources, and data schema. At the third step, we develop Big Data Generator Suite (in short, BDGS)to generate synthetic big data, including text, graph, and table data. At the fourth step, since there are different lines of systems, for same workloads, different implementations should be available for performing an apples-to-apples comparison of different systems. The fifth step ensures our workloads are representative in terms of not only workloads and data characteristics, but also system characteristics, you can find more details on the BigDataBench-Lite Item. After five steps, finally, we decide our big data analytics benchmarks.

3.2 Chosen Data sets and Workloads

Covering three application scenarios, BigDataBench3.0 includes six real-world data sets, two synthetic data sets and 32 big data workloads.

Table 1: The Summary of Data Sets

No.	data sets	data size
1	Wikipedia Entries	4,300,000 English articles
2	Amazon Movie Reviews	7,911,684 reviews
3	Google Web Graph	875713 nodes, 5105039 edges
4	Facebook Social Network	4039 nodes, 88234 edges
5	E-commerce Transaction Data	table1: 4 columns, 38658 rows.table2: 6 columns, 242735 rows
6	ProfSearch Person Resumes	278956 resumes
7	CALDA Table Data(synthetic data) https://issues.apache.org/jira/i#browse/HIVE-396	table1: 3 columns, table2:9 columns.
8	TPC-DS WebTable Data(synthetic data) http://www.tpc.org/tpcds/	26 tables

Table 2: The Summary of BigDataBench

Application Scenarios	Application Type	Workloads	Data Sets	Software Stacks
Cloud OLTP	Micro Benchmarks	Read	Semi-structured Table	HBase, Mysql
		Write		
		Scan		
	Applications	Search Server	Semi-structured Table	HBase, Nutch
	Micro Benchmarks	Sort	Unstructured Text	MPI, Spark, Hadoop
		Grep		
		WordCount		
		BFS	Unstructured Graph	MPI
		Index	Unstructured Text	
		PageRank	Unstructured Graph	
		Kmeans	Unstructured	
		Connected Components		

Offline Analytics	Analytics Workloads		Graph	MPI, Spark, Hadoop
		Collaborative Filtering	Semi-structured Text	
		Naive Bayes		
OLAP and Interactive Analytics	Micro Benchmarks	Project	structured Table	Mysql, Hive, Shark, Impala
		Filter		
		OrderBy		
		Cross Product		
		Union		
		Difference		
		Aggregation		
	Analytics Workloads	Join Query		
		Select Query		
		Aggregation Query		
		Eight TPC-DS Web Queries		

3.3 Use case

This subsection will describe the application example of BigDataBench.

General procedures of using BigDataBench are as follows:
1. Choose the proper workloads. Select workloads with the specified purpose, for example basic operations in Hadoop environment or typical search engine workloads.

2. Prepare the environment for the corresponding workloads. Before running the experiments, the environment should be prepared first for example the Hadoop environment.

3. Prepare the needed data for workloads. Generally, it's necessary to generate data for the experiments with the data generation tools.

4. Run the corresponding applications. With all preparations done, it's needed to start the workload applications and/or performance monitoring tools in this step.

5. Collect results of the experiments.

Here we provide two usage case to show how to use our benchmark to achieve different evaluating task.
Case one: from the perspective of choosing suitable big data system
If the data center administrator of an e-commerce company wants to choose a suitable big data management system to conduct business intelligence analysis and decision support, and they aim

to find big data system with shorter response time. The application scenarios mainly include two types: interactive analytics queries and OLAP queries, and the data scale is known according to their application characteristics. First he should choose the suitable workloads: the micro benchmarks which include basic operations in interactive data analytics, the three interactive analytics workloads (include select query, aggregation query and join query), and the OLAP workloads including 8 TPC-DS queries. Next, he should prepare the data sets of different data sizes using BDGS(Big Data Generator Suite) provided in BigDataBench, and choose the execution model according to the hardware configuration (tables-in-memory query or tables-on-disk query). Finally, he can run the selected workloads and make decision according to evaluation results.

Case two: from the perspective of architecture

Suppose that someone is planning to design a new machine for common big data usage. It is not enough to run subset of the workloads, since he doesn't know what special application scene and soft stack the new machine is used for. The comprehensive evaluation is needed, so that he should run every workload to reflect the performance of different application scene, program framework, data warehouse, and NoSQL database. Only in this way, he can say his new design is indeed beneficial for big data usage.

Other use cases of BigDataBench include:

Data Analysis workload's feature:

Another kind of use case is to observe the typical data analysis workloads'(for example PageRank, Recommendation) architectural characters.

Different storage system:

In BigDataBench, we also provide different data management systems(for example HBase, Cassandra, hive). Users can choose one or some of them to observe the architectural feature by running the basic operations(sort, grep, wordcount).

Different programing models:

Users can use BigDataBench to study three different programing models: MPI, MapReduce and Spark.

4 Prerequisite Software Packages

Software	Version	Download
Hadoop	1.0.2	http://hadoop.apache.org/#Download+Hadoop
HBase	0.94.5	http://www.apache.org/dyn/closer.cgi/hbase/
Cassandra	1.2.3	http://cassandra.apache.org/download/
MongoDB	2.4.1	http://www.mongodb.org/downloads
Mahout	0.8	https://cwiki.apache.org/confluence/display/MAHOUT/Downloads
Hive	0.9.0	https://cwiki.apache.org/confluence/display/Hive/GettingStarted #GettingStarted-InstallationandConfiguration
Spark	0.8.0	http://spark.incubator.apache.org/
Impala	1.1.1	http://www.cloudera.com/content/cloudera-content/ cloudera-docs/Impala/latest/Installing-and-Using-Impala/ciiu_install.html
MPICH	2.0	http://www.mpich.org/downloads/
Boost	1_43_0	http://www.boost.org/doc/libs/1_43_0/more/getting_started/unix-variants.html
Shark	0.8.0	http://shark.cs.berkeley.edu/
Scala	2.9.3	http://www.scala-lang.org/download/2.9.3.html
GCC	4.8.2	http://gcc.gnu.org/releases.html
GSL	1.16	http://www.gnu.org/software/gsl/

5 Big Data Generator Suite

In BigDataBench 3.0, we introduce Big Data Generator Suite, a comprehensive tool developed to generate synthetic big data preserving the 4V properties. Specifically, our BDGS can generate data using a sequence of steps. First, BDGS selects application-specific and representative real-world data sets. Secondly, 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 is consist of three parts: Text generator, Graph generator, and Table generator. We will introduce how to use these tools to generate data as following.

5.1 Text Generator

We provide a data generation tool which can generate data with user specified data scale. In BigDataBench 3.0 we analyze the wiki data sets to generate model, and our text data generate tool

can produce the big data based on the model.

Usage

Generate the data

Basic command-line usage:

sh gen_text_data.sh MODEL_NAME FIlE_NUM FILE_LINES LINE_WORDS OUT_DATA_DIR

< MODEL_NAME >: the name of model used to generate new data

< FIlE_NUM >: the number of files to generate

< FILE_LINES >: number of lines in each file

< LINE_WORDS >: number of words in each line

<OUT_DATA_DIR >: output director

For example:

sh gen_text_data.sh lda_wiki1w 10 100 1000 gen_data/

This command will generate 10 files each contains 100 lines, and each contains 1000 words by using model wiki1w.

Note: The tool Need to install GSL - GNU Scientific Library. Before you run the program, Please make sure that GSL is ready.

Your also can choose parallel ,it runs like this:

mkdir /mnt/raid/BigDataGeneratorSuite in every node

Configure Non password login and the host: parallel_ex/conf_hosts

To Run:

cd parallel_ex/

sh deploy_ex.sh

sh run_textGen_.sh

5.2 Graph Generator

Here we use Kronecker to generate data that is both mathematically tractable and have all the structural properties from the real data set. (http://snap.stanford.edu/snap/index.html)

In BigDataBench 3.0X we analyze the Google, Facebook and Amazon data sets to generate model, and our graph data generate tool can produce the big data based on the model.

Usage

Generate the data

Basic command-line usage:

./ gen_kronecker_graph

-o:Output graph file name (default:'graph.txt')

-m:Matrix (in Maltab notation) (default:'0.9 0.5; 0.5 0.1')

-i:Iterations of Kronecker product (default:5)
-s:Random seed (0 - time seed) (default:0)

For example:

./ gen_kronecker_graph -o:../data-outfile/amazon_gen.txt -m:"0.7196 0.6313; 0.4833 0.3601" -i:23

5.3 Table Generator

We use Parallel Data Generation Framework to generate table data. The Parallel Data Generation Framework (PDGF) is a generic data generator for database benchmarking. PDGF was designed to take advantage of today's multi-core processors and large clusters of computers to generate large amounts of synthetic benchmark data very fast. PDGF uses a fully computational approach and is a pure Java implementation which makes it very portable.
You can use your own configuration file to generate table data.

Usage

1. **Prepare the configuration files**

The configuration files are written in XML and are by default stored in the *config* folder. PDGF-V2 is configured with 2 XML files: the schema configuration and the generation configuration. The schema configuration (demo-schema.xml) defines the structure of the data and the generation rules, while the generation configuration (demo-generation.xml) defines the output and the post-processing of the generated data.
For the demo, we will generate the files demo-schema.xml and demo-generation.xml which are also contained in the provided .gz file. Initially, we will generate two tables: OS_ORDER and OS_ORDER_ITEM.

- demo-schema.xml
- demo-generation.xml

2. **Generate data**

After creating both demo-schema.xml and demo-generation.xml a first data generation run can be performed. Therefore it is necessary to open a shell, change into the PDGF Environment directory.

Basic command-line usage WITH Scale Factor:

cd Table_datagen/e-com
java -XX:NewRatio=1 -jar pdgf.jar -l demo-schema.xml -l demo-generation.xml -c -s -sf 2000

Your also can choose parallel ,it runs like this:

mkdir /mnt/raid/BigDataGeneratorSuite in every node
Configure Non password login and the host: parallel_ex/conf_hosts

To Run:

cd parallel_ex/
sh deploy_ex.sh

sh run_personalResumeGen.sh

6 Workloads

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 introduction how to run the Big Data Benchmark for each workload. It mainly has two steps. The first step is to generate the big data and the second step is to run the applications using the data we generated.

6.1 Cloud OLTP

6.1.1 MicroBenchmarks

We use YCSB to run database basic operations. And, we provide three ways: HBase, Cassandra and MongoDB to run operations for each operation.

To Prepare
Obtain YCSB:
wget https://github.com/downloads/brianfrankcooper/YCSB/ycsb-0.1.4.tar.gz
tar zxvf ycsb-0.1.4.tar.gz
cd ycsb-0.1.4
Or clone the git repository and build:
git clone git://github.com/brianfrankcooper/YCSB.git
cd YCSB
mvn clean package
We name $YCSB as the path of YCSB for the following steps.

Write:

1. **For HBase**

Basic command-line usage:

cd $YCSB
sh bin/ycsb load hbase -P workloads/workloadc -p threads=<thread-numbers> -p columnfamily=<family> -p recordcount=<recordcount-value> -p hosts=<hostip> -s>load.dat

A few notes about this command:

- <thread-number> : the number of client threads, this is often done to increase the amount of load offered against the database.
- <family> : In Hbase case, we used it to set database column. You should have database usertable with column family before running this command. Then all data will be loaded into database usertable with column family.
- <recorcount-value>: the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.

- <hostip> : the IP of the hbase's master node.

2. For Cassandra

Before you run the benchmark, you should create the keyspace and column family in the Cassandra. You can use the following commands to create it:

```
CREATE KEYSPACE usertable
  with placement_strategy = 'org.apache.cassandra.locator.SimpleStrategy'
and strategy_options = {replication_factor:2};
use usertable;
create column family data with comparator=UTF8Type and
default_validation_class=UTF8Type and key_validation_class=UTF8Type;
```

Basic command-line usage:

```
cd $YCSB
sh bin/ycsb load cassandra-10  -P  workloads/workloadc  -p threads=<thread-numbers>
-p recordcount=<recorcount-value> -p  hosts=<hostips>  -s >load.dat
```

A few notes about this command:

- <thread-number> : the number of client threads, this is often done to increase the amount of load offered against the database.
- <recorcount-value> : the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
- <hostips> : the IP of cassandra's nodes. If you have more than one node you should divide with ",".

3. For MongoDB

Basic command-line usage:

```
cd $YCSB
/bin/ycsb load mongodb  -P workloads/workloadc -p threads=<thread-numbers> -p
recordcount=<recorcount-value> -p mongodb.url=<mongodb.url> -p
mongodb.database=<database> -p mongodb.writeConcern=normal  -s >load.dat
```

A few notes about this command:

- <thread-number> : the number of client threads, this is often done to increase the amount of load offered against the database.
- <recorcount-value>: the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
- <mongodb.url>: this parameter should point to the mongos of the mongodb. For example "mongodb://172.16.48.206:30000".
- <database>: In Mongodb case, we used it to set database column. You should have database ycsb with collection usertable before running this command. Then all data will be loaded into database ycsb with collection usertable. To create the database and the collection, you can use the following commands:

```
db.runCommand({enablesharding:"ycsb"});
db.runCommand({shardcollection:"ycsb.usertable",key:{ _id:1}});
```

Read:

1. For HBase

Basic command-line usage:

```
cd $YCSB
sh bin/ycsb run hbase -P workloads/workloadc -p threads=<thread-numbers> -p
columnfamily=<family> -p operationcount=<operationcount-value> -p hosts=<hostip>
-s>tran.dat
```

A few notes about this command:

- <thread-number> : the number of client threads, this is often done to increase the amount of load offered against the database.
- <family> : In Hbase case, we used it to set database column. You should have database usertable with column family before running this command. Then all data will be loaded into database usertable with column family.
- < operationcount-value >: the total operations for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
- <hostip> : the IP of the hbase's master node.

2. For Cassandra

Basic command-line usage:

```
cd $YCSB
sh bin/ycsb run cassandra-10 -P workloads/workloadc -p threads=<thread-numbers> -p
operationcount=<operationcount-value> -p hosts=<hostips> -s>tran.dat
```

A few notes about this command:

- <thread-number> : the number of client threads, this is often done to increase the amount of load offered against the database.
- <operationcount-value> : the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
- <hostips> : the IP of cassandra's nodes. If you have more than one node you should divide with ",".

3. For MongoDB

Basic command-line usage:

```
cd $YCSB
sh bin/ycsb run mongodb -P workloads/workloadc -p threads=<thread-numbers> -p
operationcount=<operationcount-value> -p mongodb.url=<mongodb.url> -p
mongodb.database=<database> -p mongodb.writeConcern=normal -p
mongodb.maxconnections=<maxconnections> -s>tran.dat
```

A few notes about this command:

- <thread-number> : the number of client threads, this is often done to increase the amount of load offered against the database.
- <operationcount-value> : the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
- <mongodb.url>: this parameter should point to the mongos of the mongodb. For example "mongodb://172.16.48.206:30000".

- <database>: In Mongodb case, we used it to set database column. You should have database ycsb with collection usertable before running this command. Then all data will be loaded into database ycsb with collection usertable. To create the database and the collection, you can use the following commands:
  db.runCommand({enablesharding:"ycsb"});
  db.runCommand({shardcollection:"ycsb.usertable",key:{ _id:1}});
- <maxconnections> : the number of the max connections of mongodb.

Scan:

1. For HBase

Basic command-line usage:

cd $YCSB

sh bin/ycsb run hbase -P workloads/workloade -p threads=<thread-numbers> -p columnfamily=<family> -p operationcount=<operationcount-value> -p hosts=<Hostip> -p columnfamily=<family> -s>tran.dat

A few notes about this command:

- <thread-number> : the number of client threads, this is often done to increase the amount of load offered against the database.
- <family> : In Hbase case, we used it to set database column. You should have database usertable with column family before running this command. Then all data will be loaded into database usertable with column family.
- < operationcount-value >: the total operations for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
- <hostip> : the IP of the hbase's master node.

2. For Cassandra

Basic command-line usage:

cd $YCSB

sh bin/ycsb run cassandra-10 -P workloads/workloade -p threads=<thread-numbers> -p operationcount=<operationcount-value> -p hosts=<hostips> -s>tran.dat

A few notes about this command:

- <thread-number> : the number of client threads, this is often done to increase the amount of load offered against the database.
- <operationcount-value> : the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
- <hostips> : the IP of cassandra's nodes. If you have more than one node you should divide with ",".

3. For MongoDB

Basic command-line usage:

cd $YCSB

sh bin/ycsb run mongodb -P workloads/workloade -p threads=<thread-numbers> -p operationcount=<operationcount-value> -p mongodb.url=<mongodb.url> -p mongodb.database=<database> -p mongodb.writeConcern=normal -p

mongodb.maxconnections=<maxconnections> -s>tran.dat

A few notes about this command:

- <thread-number> : the number of client threads, this is often done to increase the amount of load offered against the database.
- <operationcount-value> : the total records for this benchmark. For example, when you want to load 10GB data you shout set it to 10000000.
- <mongodb.url>: this parameter should point to the mongos of the mongodb. For example "mongodb://172.16.48.206:30000".
- <database>: In Mongodb case, we used it to set database column. You should have database ycsb with collection usertable before running this command. Then all data will be loaded into database ycsb with collection usertable. To create the database and the collection, you can use the following commands:

  db.runCommand({enablesharding:"ycsb"});

  db.runCommand({shardcollection:"ycsb.usertable",key:{ _id:1}});

<maxconnections> : the number of the max connections of mongodb.

6.2 Offline Analytics

6.2.1 MicroBenchmarks

1. Hadoop-version (sort, grep, wordcount)

To prepare:

1. Please decompress the file: BigDataBench_V 3.0.tar.gz

   tar xzf BigDataBench_V3.0.tar.gz
2. Open the DIR:

   cd BigDataBench_V3.0_Hadoop_Hive /MicroBenchmarks/
3. Gnerate data

   sh genData_MicroBenchmarks.sh

To run wordcount and grep you can only do like this:

sh run_MicroBenchmarks.sh

but when you choose to run sort you should put the sort-transfer file in your hadoop and then to run like this(the sort-transfer you can find in BigDataBench_V3.0.tar.gz)

first : sh genData_MicroBenchmarks.sh

second : sh sort-transfer.sh

last : sh run_MicroBenchmarks.sh (choose sort to run)

2. Spark-version (sort, grep, wordcount)

(If you use not one machine you must download the spark on each machines,and must download in the right way**)**

To prepare:

1. Please decompress the file: BigDataBench_Sprak_V3.0.tar.gz

   tar xzf BigDataBench_Sprak_V3.0.tar.gz

2. Open the DIR:

cd BigDataBench_V3.0_Spark+Shark/MicroBenchmarks/

3. Gnerate data

sh genData_MicroBenchmarks.sh

sh sort-transfer.sh

To run:

when you chose sort like this:

before to run the sort you should put the sort-transfer file in your hadoop and then to run like this:

(the sort-transfer you can find in BigDataBench_Sprak_V3.0.tar)

first : sh genData_MicroBenchmarks.sh

second : sh sort-transfer.sh

last:

./run-bigdatabench cn.ac.ict.bigdatabench.Sort <master> <data_file> <save_file> [<slices>]

parameters:

<master>: URL of Spark server, for example: spark://172.16.1.39:7077

<data_file>: the HDFS path of input data, for example: /test/data.txt

<save_file>: the HDFS path to save the result

[<slices>]: optional, times of number of workers

when you chose grep like this:

./run-bigdatabench cn.ac.ict.bigdatabench.Grep <master> <data_file> <keyword> <save_file> [<slices>]

parameters:

<master>: URL of Spark server, for example: spark://172.16.1.39:7077

<data_file>: the HDFS path of input data, for example: /test/data.txt

<keyword>: the keyword to filter the text

<save_file>: the HDFS path to save the result

[<slices>]: optional, times of number of workers

When you chose wordcount lie this:

./run-bigdatabench cn.ac.ict.bigdatabench.WordCount <master> <data_file> <save_file> [<slices>]

parameters:

<master>: URL of Spark server, for example: spark://172.16.1.39:7077

<data_file>: the HDFS path of input data, for example: /test/data.txt

<save_file>: the HDFS path to save the result

[<slices>]: optional, times of number of workers

3. Mpi-version (sort, grep, wordcount)

(If you use not one machine you must put the MPI on each mpi-machines and put in the same path**)**

1) Sort:

To prepare:

1. Please decompress the file: BigDataBench_MPI_V3.0.tar.gz

tar xzf BigDataBench_MPI_V3.0.tar.gz

2. Open the DIR:

   cd BigDataBench_V3.0_MPI/MicroBenchmarks/MPI_Sort/

3. Gnerate data

   sh genData_sort.sh

To makefile:

This we provid two version,you can choose make it by yourself ,if you do that you must translate like this:

make

And you also can use mpi_sort ,we have already translated directly, the translated file is run_sort

To run:

mpirun -n process_number ./mpi_sort <hdfs Path> <hdfs port> <input_file> <output_file>

For example:

mpirun -n 24 ./mpi_sort 172.18.11.107 9000 /home/mpi /data

2) Grep:

To prepare:

1. Please decompress the file: BigDataBench_MPI_V3.0tar.gz

   tar xzf BigDataBench_MPI_V3.0tar.gz

2. Open the DIR:

   cd BigDataBench_MPI_V3.0tar.gz /MicroBenchmarks/ MPI_Grep/

3. Gnerate data

   sh genData_grep.sh

Then there will be a data-grep file in the current directory,you can find your datas in it .If you use not one machine you must put the datas on each mpi-machines,most of all you must pur them in the same path .

To makefile:

This we provid two version,you can choose make it by yourself ,if you do that you must translate like this :

make

And you also can use mpi_grep ,we have already translated directly, the translated file is run_grep

To run:

mpirun -n process_number ./mpi_grep [input_file] [pattern]

3) Wordcount:

To prepare:

1. Please decompress the file: BigDataBench_MPI_V3.0.tar.gz

   tar xzf BigDataBench_MPI_V3.0tar.gz

2. Open the DIR:

   cdBigDataBench_MPI_V3.0.tar.gz /MicroBenchmarks/ MPI_WordCount/

3. Gnerate data

   sh genData_wordcount.sh

Then there will be a data-wordcount file in the current directory,you can find your datas in it .If you use not one machine you must put the datas on each mpi-machines,most of all you must pur them in the same path .

To makefile:

This we provid two version,you can choose make it by yourself ,if you do that you must translate like this :

make

And you also can use mpi_wordcount, we have already translated directly, the translated file is mpi_wordcount

To run:

wordcount like this:

mpirun -n process_number ./run_wordcount <input_file>

4. BFS (Breath first search)

To prepare

4. Please decompress the file: BigDataBench_V3.0.tar.gz
   tar xzf BigDataBench_V3.0.tar.gz
5. Open the DIR: graph500
   cd BigDataBench_V3.0/MicroBenchmarks/BFS/graph500

Basic Command-line Usage:

mpirun -np PROCESS_NUM graph500/mpi/graph500_mpi_simple VERTEX_SIZE

Parameters:
PROCESS_NUM: number of process;
VERTEX_SIZE: number of vertex, the total number of vertex is 2^ VERTEX_SIZE

For example: Set the number of total running process of to be 4, the vertex number to be 2^20, the command is:
mpirun -np 4 graph500/mpi/graph500_mpi_simple 20

6.2.2 Analytics Workloads

PageRank:

The PageRank program now we use is obtained from Hibench.

1. Hadoop-version

To prepare and generate data:

tar xzf BigDataBench_V3.0.tar.gz
cd BigDataBench_V3.0_Hadoop_Hive/SearchEngine/PageRank
sh genData_PageRank.sh

To run:

sh run_PageRank.sh [#_of_nodes] [#_of_reducers] [makesym or nosym] [#_Iterations_of_GenGragh]

[#_of_nodes] : number of nodes in the graph
[#_of_reducers] : number of reducers to use in hadoop. - The number of reducers to use depends on the setting of the hadoop cluster. - The rule of thumb is to use (number_of_machine * 2) as the number of reducers.
[#_Iterations_of_GenGragh] : Iterations of GenGragh

2. Spark-version

(If you use not one machine you must download the spark on each machines,and must download in the right way**)**

To prepare:

1. Please decompress the file: BigDataBench_Sprak_V3.0.tar.gz
   tar xzf BigDataBench_Sprak_V3.0tar.gz
2. Open the DIR:
   cd BigDataBench_Sprak_V3.0 /SearchEngine/ Pagerank
3. Gnerate data
   sh genData_PageRank.sh

To run:

./run-bigdatabenchorg.apache.spark.examples.PageRank
<master> <file> <number_of_iterations> <save_path> [<slices>]
parameters:
#<master>: URL of Spark server, for example: spark://172.16.1.39:7077
#<file>: the HDFS path of input data, for example: /test/data.txt
#<number_of_iterations>: number of iterations to run the algorithm
#<save_path>: path to save the result
#[<slices>]: optional, times of number of workers

3. Mpi-version

(If you use not one machine you must put the MPI on each mpi-machines and put in the same path**)**

To prepare:

1. Please decompress the file: BigDataBench_MPI_V3.0tar.gz
   tar xzf BigDataBench_MPI_V3.0.tar.gz
2. Open the DIR:
   cd BigDataBench_MPI_V3.0 / SearchEngine / MPI_Pagerank
3. Gnerate data
   sh genData_ PageRank.sh

Then there will be a data- PageRank file in the current directory,you can find your datas in it .If you use not one machine you must put the datas on each mpi-machines,most of all you must pur them in the same path .

To makefile:

This we provid two version,you can choose make it by yourself ,if you do that you must translate like this :

1 Install boost and cmake

2cd/BigDataBench_V3.0_MPI/SearchEngine/MPI_Pagerank/parallel-bgl-0.7.0/libs/graph_parallel/test

make distributed_page_rank_test

And you also can use run_PageRank, we have already translated directly, the translated file is run_PageRank

To run:

mpirun -n process_number ./run_PageRank <InputGraphfile> <num_ofVertex> <num_ofEdges> <iterations>

parameters:

#<num_ofVertex> <num_ofEdges> these two parameters you can find in your gen_data

<num_ofEdges>: data length as L

<num_ofVertex>: 2^n

<iterations>: n

Index:

The Index program now we use is obtained from Hibench.

To prepare:

tar xzf BigDataBench_V3.0.tar.gz

cd BigDataBench_V3.0_Hadoop_Hive/SearchEngine/Index

(when you do prepare.sh you must put linux.words and words these two files in /usr/share/dict)

sh prepare.sh

Basic command-line usage:

sh run_Index.sh

Kmeans:

The Kmeans program we used is obtained from Mahout.

1. Hadoop-version

To Prepare

tar xzf BigDataBench_V3.0.tar.gz

cd BigDataBench_V3.0_Hadoop_Hive /SNS/Kmeans

sh genData_Kmeans.sh

Basic command-line usage:

sh run_Kmeans.sh

2. Spark-version

(If you use not one machine you must download the spark on each machines,and must download in the right way**)**

To prepare:

1. Please decompress the file: BigDataBench_Sprak_V3.0.tar.gz
   tar xzf BigDataBench_Sprak_V3.0.tar.gz
4. Open the DIR:

cd BigDataBench_V3.0_Spark+Shark /SNS /Kmeans

5. Gnerate data
   sh genData_ Kmeans.sh

To run:

./run-bigdatabench org.apache.spark.mllib.clustering.KMeans <master> <input_file> <k> <max_iterations> [<runs>]

parameters:
#<master>: URL of Spark server, for example: spark://172.16.1.39:7077
#<input_file>: the HDFS path of input data, for example: /test/data.txt
#[<k>]: number of centers
#<max_iterations>: number of iterations to run the algorithm
#[<runs>]: optional, level of parallelism to split computation into

3. Mpi-version

(If you use not one machine you must put the MPI on each mpi-machines and put in the same path**)**

Simple-kmeans：

To prepare:

1. Please decompress the file: BigDataBench_MPI_V3.0.tar.gz

tar xzf BigDataBench_MPI_V3.0.tar.gz

2. Open the DIR:

cd BigDataBench_MPI_V3.0/ SNS / Simple_Kmeans

./Generating Image_data/color100.txt 100000 > wl.1

The number of 100000 represent output grequency,and the number of outbound must more then the number of wl.1 .

To makefile:

This we provid two version,you can choose make it by yourself ,if you do that you must translate like this

mpicxx Generating.cpp -o mpi_main

And you also can use run we have already translated directly, the translated file is mpi_main

To run:

mpirun -np 12 ./mpi_main -i wl.1 -n 10 –o
then you will get a new cluster file like wl.1.

parameters:
-i:the data set of clusters
-n:the number of clusters like kmeans'K
-o:output file
Then there will be a data in the current directory..If you use not one machine you must put the datas on each mpi-machines,most of all you must pur them in the same path .

Connected Components:

The Connected Components program we used is obtained from PEGASUS.

1. Hadoop-version

To Prepare

tar xzf BigDataBench_V3.0.tar.gz
cd BigDataBench_V3.0_Hadoop_Hive /SNS/Connected_Components
sh genData_connectedComponents.sh

Basic command-line usage:

sh run_connectedComponents.sh [#_of_nodes] [#_of_reducers] [#_Iterations_of_GenGragh]

[#_of_nodes] : number of nodes in the graph
[#_of_reducers] : number of reducers to use in hadoop. - The number of reducers to use depends on the setting of the hadoop cluster. - The rule of thumb is to use (number_of_machine * 2) as the number of reducers.
[#_Iterations_of_GenGragh] : Iterations of GenGragh

2. Spark-version

(If you use not one machine you must download the spark on each machines,and must download in the right way**)**

To prepare:

1. Please decompress the file: BigDataBench_Sprak_V3.0.tar.gz
   tar xzf BigDataBench_Sprak_V3.0.tar.gz
2. Open the DIR:
   cd BigDataBench_V3.0_Spark+Shark / SNS/connect/
3. Gnerate data
   sh genData_ connectedComponents.sh

To run:

./run-bigdatabench cn.ac.ict.bigdatabench.ConnectedComponent <master> <data_file> [<slices>]
parameters:
#<master>: URL of Spark server, for example: spark://172.16.1.39:7077
#<data_file>: the HDFS path of input data, for example: /test/data.txt

#[<slices>]: optional, times of number of workers

3. Mpi-version

(If you use not one machine you must put the MPI on each mpi-machines and put in the same path**)**

To prepare:

1. Please decompress the file: BigDataBench_MPI_V3.0.tar.gz
   tar xzf BigDataBench_MPI_V3.0.tar.gz
2. Open the DIR:
   cd BigDataBench_MPI_V3.0 / SNS/MPI_Connect
4. Gnerate data
   sh genData_connectedComponents.sh

Then there will be a data- Connected_Components file in the current directory,you can find your datas in it .If you use not one machine you must put the datas on each mpi-machines,most of all you must pur them in the same path .

To makefile:

This we provid two version,you can choose make it by yourself ,if you do that you must translate like this :

1 Install boost and cmake

2cd/BigDataBench_V3.0_MPI/SNS/Connected_Components/parallel-bgl-0.7.0/
libs/graph_parallel/test

make distributed_ramt_cc

And you also can use run_connectedComponents, we have already translated directly, the translated file is run_connectedComponents

To run:

mpirun -n process_number ./run_connectedComponents <InputGraphfile> <num_ofVertex> <num_ofEdges>

parameters:

<num_ofVertex> <num_ofEdges> these two parameters you can find in your gen_data

<num_ofEdges>: data length as L

<num_ofVertex>: 2^n

Collaborative Filtering Recommendation:

Collaborative filtering recommendation is one of the most widely used algorithms in E-com analysis. It aims to solve the prediction problem where the task is to estimate the preference of a user towards an item which he/she has not yet seen.

We use the RecommenderJob in Mahout(http://mahout.apache.org/) as our Recommendation workload, which is a completely distributed itembased recommender. It expects ID1, ID2, value (optional) as inputs, and outputs ID1s with associated recommended ID2s and their scores. As you

know, the data set is a kind of graph data.

Before you run the RecommenderJob, you must have HADOOP and MAHOUT prepared. You can use Kronecker (see 4.2.1) to generate graph data for RecommenderJob.

Basic command-line usage:

```
tar xzf BigDataBench_V3.0.tar.gz
cd BigDataBench_V3.0_Hadoop_Hive/E-commerce
sh genData_recommendator.sh
sh run_recommendator.sh
```

Input parameters according to the instructions:

1. The DIR of your working director;
2. The DIR of MAHOUT_HOME.

Naive Bayes

Naive Bayes is an algorithm that can be used to classify objects into usually binary categories. It is one of the most common learning algorithms in Classification. Despite its simplicity and rather naive assumptions it has proven to work surprisingly well in practice.

We use the naivebayes in Mahout(http://mahout.apache.org/) as our Bayes workload, which is a completely distributed classifier.
When you choose to run bayes,we should use mahout-0.6 .So we provide the mahout-0.6 in E-commerce.You must install and export the environment.You can do like this:

```
cd BigDataBench_V3.0/E-commerce
tar -zxvf mahout-distribution-0.6.tar.gz
export BigDataBench_V3.0/E-commerce/ mahout-distribution-0.6
```

and then you can run it

1. Hadoop-version

Basic command-line usage:

```
tar xzf BigDataBench_V3.0.tar.gz
cd BigDataBench_V3.0_Hadoop_Hive /E-commerce
sh genData_naivebayes.sh
sh run_naivebayes.sh
```

Input parameters according to the instructions:

The DIR of your working director;
The DIR of MAHOUT_HOME.

2. Spark-version

(If you use not one machine you must download the spark on each machines,and must download in the right way**)**

To prepare:

1. Please decompress the file: BigDataBench_Sprak_V3.0.tar.gz
   tar xzf BigDataBench_Sprak_V3.0.tar.gz
5. Open the DIR:
   cd BigDataBench_V3.0_Spark+Shark / E-commerce
6. Gnerate data
   sh genData_ naivebayes.sh

To run:

sh run_naivebayes.sh

3. Mpi-version

(If you use not one machine you must put the MPI on each mpi-machines and put in the same path**)**

To prepare:

1. Please decompress the file:
   tar xzf BigDataBench_MPI_V3.0.tar.gz
2. Open the DIR:
   cd BigDataBench_MPI_V3.0 / E-commerce/MPI_naivebayes
7. Gnerate data
   sh genData_naivebayes.sh

The naivebayes is special ,you can not copy the data to each machines .You must use everyone machine's genData_naivebayes.sh to generate datas,and then you can change prcoess numberes.

To makefile:

This we provid two version,you can choose make it by yourself ,if you do that you must translate like this

mpic++ -std=c++11 -o MPI_NB MPI_NB.cpp

And you also can use run_sort we have already translated directly, the translated file is run_naivebayes

To run:

mpirun -n process_number ./run_naivebayes -i <inputfile> -o <save_file>

(the bayes algorithm need we use BigdataSuit in every machines. We must genearte datas fisrt.Not like others to copy the datas to every machines.)

6.3 OLAP and Interactive Analytics

6.3.1 MicroBenchmarks

1. Hive Version

To prepare and generate data: With reference to 5.3 Table Generator

Create tables and load data into tables:

(MicroBenchmark and AMPLabWorkloads use the same table)

cd $HIVE_HOME/bin
./hive
create database bigdatabench;
use bigdatabench;
create table bigdatabench_dw_order(order_id int,buyer_id int,create_date string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE;

create table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE;
load data local inpath
'$BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/OS_ORDER.txt'
overwrite into table bigdatabench_dw_order;
load data local inpath '$BigDataBench_HOME
/BigDataGeneratorSuite/Table_datagen/output/OS_ORDER_ITEM.txt' overwrite into table
bigdatabench_dw_item;
create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run:

cd Interactive_MicroBenchmark
sh run-MicroBenchmark.sh
(For ease of use, we recommend that you use a local mysql server to store metadata)

2. Shark Version

To prepare and generate data:

With reference to 5.3 Table Generator

Upload the text files in $BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/ to HDFS and make sure these files in different pathes.

Create tables:

tar zxvf MicroBenchmark.tar
cd Interactive_MicroBenchmark
shark
create external table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION ‘path to OS_ODER_ITEM.txt’;
create external table bigdatabench_dw_order(order_id int,buyer_id int,create_date string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION ‘path to OS_ORDER.txt’;
create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run:

cd Interactive_MicroBenchmark
edit free_m.sh to make sure it runs correctly.

sh runMicroBenchmark.sh

3. Impala Version

To prepare and generate data:

With reference to 5.3 Table Generator

Upload the text files in $BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/ to HDFS and make sure these files in different pathes

Create tables:

$HIVE_HOME/bin

./hive

create external table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION ‘path to OS_ODER_ITEM.txt’;

create external table bigdatabench_dw_order(order_id int,buyer_id int,create_date string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION ‘path to OS_ORDER.txt’;

create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run:

tar zxvf MicroBenchmark.tar.gz

cd MicroBenchmark

edit free_m.sh and impala-restart.sh to make sure them run correctly.

sh runMicroBenchmark.sh

6.3.2 Analysitic workloads

1. Hive Version

AMPLab workloads

To prepare and generate data:

the data of table bigdatabench_dw_order and talbe bigdatabench_dw_item：

With reference to the previous section

Create tables and load data into tables:

(MicroBenchmark and AMPLabWorkloads use the same table)

cd $HIVE_HOME/bin

./hive

create database bigdatabench;

use bigdatabench;

create table bigdatabench_dw_order(order_id int,buyer_id int,create_date string)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE;

create table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE;

load data local inpath ' /home/output /OS_ORDER.txt' overwrite into table bigdatabench_dw_order;
load data local inpath ' /home/L/BigDataBench_V3.0_Hadoop_Hive/BigDataGeneratorSuite/Table_datagen/e-com output/OS_ORDER_ITEM.txt' overwrite into table bigdatabench_dw_item;
create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run:

cd Interactive_Query
sh run-AnalyticsWorkload.sh

TPC-DS workloads

To prepare and generate data:

cd OLAP_Benchmark

./dsdgen -scale <data-size> -dir <data-directory>

A few notes about this command:

- <data-size> : the number of the data size, the unite is GB.
- <data-directory> :the path where your data want to put.

./mvdata.sh <data-directory>

Create the TPC-ds tables:

Firstly, you should put TPC-DS data into the hdfs.

cd OLAP_Benchmark
sh create-tpcds-tables.sh

To run:

cd OLAP_Benchmark
sh run-TPC-DS.sh

2. Shark Version

AMPLab workloads

To prepare and generate data:

With reference to 5.3 Table Generator

Upload the text files in $BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/ to HDFS and make sure these files in different pathes.

Create tables:

tar zxvf InteractiveQuery.tar.gz
cd InteractiveQuery
shark
create external table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION 'path to OS_ODER_ITEM.txt';
create external table bigdatabench_dw_order(order_id int,buyer_id int,create_date string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION 'path to OS_ORDER.txt';
create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run:

cd InteractiveQuery

edit free_m.sh to make sure it runs correctly.

sh runQuery.sh

TPC-DS workloads

To prepare and generate data:

tar zxvf TPC-DS.tar.gz

cd OLAP_Benchmark

./dsdgen -scale <data-size> -dir <data-directory>

A few notes about this command:

- <data-size> : the number of the data size, the unite is GB.
- <data-directory> :the path where your data want to put.

./mvdata.sh <data-directory>

Put TPC-DS data into the hdfs

Create tables:

Edit create-tpcds-tables.sql to make sure the queries use the right pathes

shark –f create-tpcds-tables.sql

To run:

cd OLAP_Benchmark

edit free_m.sh to make sure it runs correctly.

sh runTPC-DS.sh

About qurey12

Query12_2 will use query12_1's result.

3. Impala Version

AMPLab workloads

To prepare and generate data:

With reference to 5.3 Table Generator

Upload the text files in $BigDataBench_HOME/BigDataGeneratorSuite/Table_datagen/output/ to HDFS and make sure these files in different pathes.

Create tables:

$HIVE_HOME/bin/hive

create external table bigdatabench_dw_item(item_id int,order_id int,goods_id int,goods_number double,goods_price double,goods_amount double) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION 'path to OS_ODER_ITEM.txt';

create external table bigdatabench_dw_order(order_id int,buyer_id int,create_date string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' STORED AS TEXTFILE LOCATION 'path to OS_ORDER.txt';

create table item_temp as select ORDER_ID from bigdatabench_dw_item;

To run:

cd InteractiveQuery

edit free_m.sh and impala-restart.sh to make sure them run correctly.

sh runQuery.sh

TPC-DS workloads

To prepare and generate data:

tar zxvf TPC-DS.tar

cd TPC-DS

./dsdgen -scale <data-size> -dir <data-directory>

A few notes about this command:

- <data-size> : the number of the data size, the unite is GB.
- <data-directory> :the path where your data want to put.

./mvdata.sh <data-directory>

Put TPC-DS data into the hdfs

Create tables:

Edit create-tpcds-tables.sql to make sure the queries use the right pathes

hive –f create-tpcds-tables.sql

To run:

cd TPC-DS

Edit free_m.sh and impala-restart.sh to make sure them run correctly.

sh runTPC-DS.sh

About qurey12

Query12_2 will use query12_1's result

7 Reference

[1] "wikipedia," http://download.wikipedia.com/enwiki/latest/enwiki-latest-pages-articles.xml.bz2

[2] "Amazon reviews," http://snap.stanford.edu/data/web-Amazon.html

[3] "google web graph," http://snap.stanford.edu/data/web-Google.html

[4] "facebook graph," http://snap.stanford.edu/data/egonets-Facebook.html

[5] "Snap home page," http://snap.stanford.edu/snap/download.html

[6] J. Zhan, L. Zhang, N. Sun, L.Wang, Z. Jia, and C. Luo, "High volume throughput computing: Identifying and characterizing throughput oriented workloads in data centers," in Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International. IEEE, 2012, pp. 1712–1721.

[7] Zhen Jia, Lei Wang, Jianfeng Zhan, Lixin Zhang, and Chunjie Luo, Characterizing data analysis workloads in data centers, 2013 IEEE International Symposium on Workload Characterization (IISWC 2013).

[8] Yuqing Zhu, Jianfeng Zhan, ChuliangWeng, Raghunath Nambiar, Jingchao Zhang, Xingzhen Chen, and Lei Wang. Generating comprehensive big data workloads as a benchmarking framework. In The 19th International Conference on Database Systems for Advanced Applications (DASFAA 2014), 2014.

[9] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, Cheng Zhen, Gang Lu, Kent Zhan, and Bizhu Qiu. Bigdatabench: A big data benchmark suite from internet services. The 20th IEEE International

Symposium on High-Performance Computer Architecture(HPCA), 2014.
[10] Chunjie Luo, Jianfeng Zhan, Zhen Jia, Lei Wang, Gang Lu, Lixin Zhang, Chengzhong Xu, and Ninghui Sun. Cloudrank-d: benchmarking and ranking cloud computing systems for data processing applications. Frontiers of Computer Science, 6(4):347–362, 2012.