BigDataBench Technical Report

ICT, Chinese Academy of Sciences

CONTACTS (EMAIL)
PROF. JIANFENG ZHAN:
ZHANJIANFENG@ICT.AC.CN

1 Introduction

As a multi-discipline research and engineering effort, i.e., system, architecture, and data management, from both industry and academia, BigDataBench is an open-source big data benchmark suite, publicly available from `http://prof.ict.ac.cn/BigDataBench`. In nature, BigDataBench is a benchmark suite for scale-out workloads, different from SPEC CPU (sequential workloads) [16], and PARSEC (multithreaded workloads) [18]. Currently, it simulates five typical and important big data application domains: search engine, social network, e-commerce, multimedia data analytics, and bioinformatics. In specifying representative big data workloads, BigDataBench focuses on identifying units of computation, which are frequently appearing in OLTP, Cloud OLTP, OLAP, interactive and offline analytics, in each application domain. Meanwhile, it takes variety of data models into consideration, which are extracted from real-world data sets, including unstructured, semi-structured, and structured data. BigDataBench also provides an end-to-end application benchmarking framework [31] to allow the creation of flexible benchmarking scenarios by abstracting data operations and workload patterns, which can be extended to other application domains, e.g., the water consumption application domain in [8].

For the same big data benchmark specifications, different implementations are provided. For example, we and other developers implemented the offline analytics workloads using MapReduce, MPI, Spark, DataMPI, interactive analytics and OLAP workloads using Shark, Impala, and Hive. In addition to real-world data sets, BigDataBench also provides a parallel big data generation tool—BDGS—to generate scalable big data based small or medium-scale real-world data while preserving their original characteristics. The current BigDataBench version is 3.1. In total, it involves 14 real-world data sets, and 33 big data workloads.

To model and reproduce the multi-applications or multi-user scenarios on Cloud or datacenters, we provide a multi-tenancy version of BigDataBench, which allows flexible setting and replaying of mixed workloads according to the real workload traces—Facebook, Google and SoGou traces. For system and architecture researches (i. e., architecture, OS, networking and storage) the number of benchmarks will be multiplied according to different implementations, and hence become massive. To reduce the research or benchmarking cost, we select a small number of representative benchmarks, which we call the BigDataBench subset, from a large amount of BigDataBench workloads according to workload characteristics from a specific perspective. For example, for architecture communities, as simulation-based research is very time-consuming, we select a handful number of benchmarks from BigDataBench according to comprehensive micro-architectural characteristics, and provide both MARSSx86 [11] and Simics [14] simulator versions of BigDataBench.

2 Summary of BigDataBench 3.1

BigDataBench is in fast expansion and evolution. Currently, we propose several benchmark specifications to model five typical application domains. This section summarizes the implemented workloads, corresponding data sets, and scalable data generation tools. The current version BigDataBench 3.1 includes 14 real-world data sets and 33 big data workloads. Table 1 summarizes the real-world data sets and scalable data generation tools included in BigDataBench 3.1, covering the whole spectrum of data types (structured, semi-structured, and unstructured data) and different data sources, such as text, graph, image, audio, video and table data.

Table 2 presents BigDataBench from perspectives of application domains, operations/algorithms, data sets, software stacks and application types. For some end users, they may just pay attention to specified types of big data applications. For example, they want to perform an apples-to- apples comparison of software stacks for Offline Analytics. They only need to choose benchmarks with Offline Analytics. On the other hand, if the users want to measure or compare big data systems and architectures, we suggest they cover all benchmarks.

2.1 What are the differences between BigDataBench and other benchmarks?

As shown on the Table 3, among the ten desired properties, the BigDataBench is more comprehensive than other state of art big data benchmarks.

2.2 BigDataBench Evolution

As shown in Fig. 1, the evolution of BigDataBench has gone through four major stages:

At the first version, we released three benchmark suites, BigDataBench 1.0 (6 workloads from Search engine), DCBench 1.0 (11 workloads from data analytics), and CloudRank 1.0 (mixed data analytics workloads).

At the second version, we combined the previous three benchmark suites and released BigDataBench 2.0, through investigating the top three important application domains of internet services in terms of the number of page views and daily visitors. BigDataBench 2.0 is a big data benchmark suite from internet services. It includes 6 real-world data sets, and 19 big data workloads with different implementations, covering six application scenarios: micro benchmarks, Cloud OLTP, relational query, search engine, social networks, and e-commerce. Moreover, BigDataBench 2.0 provides a big data generation tool–BDGS– to generate scalable big data from small-scale real-world data while preserving their original characteristics.

In BigDataBench 3.0, we made a multidisciplinary effort to the third version, which includes 6 real-world, 2 synthetic data sets, and 32 big data workloads, covering micro and application benchmarks from typical application domains. As to generating representative and variety of big data workloads, BigDataBench

3.0 focuses on identify units of computation that frequently appear in Cloud OLTP, OLAP, interactive and offline analytics.

Now, we release the fourth version, BigDataBench 3.1. It includes 5 application domains, not only the three most important application domains from internet services, but also emerging and important domains (Multimedia analytics and Bioinformatics), altogether 14 data sets and 33 workloads. The Multi tenancy version for Cloud computing communities and simulator version for architecture communities are also released.

Table 1. The summary of data sets and data generation tools.

No.	data sets	data set description[1]	scalable data set
1	Wikipedia Entries [17]	4,300,000 English articles (unstructured text)	Text Generator of BDGS
2	Amazon Movie Reviews [7]	7,911,684 reviews (semi-structured text)	Text Generator of BDGS
3	Google Web Graph [10]	875713 nodes, 5105039 edges (unstructured graph)	Graph Generator of BDGS
4	Facebook Social Network [9]	4039 nodes, 88234 edges (unstructured graph)	Graph Generator of BDGS
5	E-commerce Transaction Data	Table 1: 4 columns, 38658 rows. Table 2: 6 columns, 242735 rows (structured table)	Table Generator of BDGS
6	ProfSearch Person Resumés	278956 resumés (semi-structured table)	Table Generator of BDGS
7	ImageNet [20]	ILSVRC2014 DET image dataset (unstructured image)	ongoing development
8	English broadcasting audio files [1]	Sampled at 16 kHz, 16-bit linear sampling (unstructured audio)	ongoing development
9	DVD Input Streams [2]	110 input streams, resolution:704*480 (unstructured video)	ongoing development
10	Image scene [3]	39 image scene description files (unstructured text)	ongoing development
11	Genome sequence data [4]	cfa data format (unstructured text)	4 volumes of data sets
12	Assembly of the human genome[5]	fa data format (unstructured text)	4 volumes of data sets
13	SoGou Data [15]	the corpus and search query data from SoGou Labs (unstructured text)	ongoing development
14	MNIST [12]	handwritten digits database which has 60,000 training examples and 10,000 test examples (unstructured image)	ongoing development

[1]The further detail of data schema is available from *BigDataBench Specification and Implementation*

Table 2. The summary of the implemented workloads in BigDataBench 3.1.

Domains	Operations or Algorithm	Types	Data Set	Software Stacks	ID[1]
Search Engine	Grep	Offline Analytics	Wikipedia Entries	MPI, Spark, Hadoop	W1-1
	WordCount	Offline Analytics	Wikipedia Entries	MPI, Spark, Hadoop	W1-2
	Index	Offline Analytics	Wikipedia Entries	MPI, Spark, Hadoop	W1-4
	PageRank	Offline Analytics	Google Web Graph	MPI, Spark, Hadoop	W1-5
	Nutch Server	Online Service	SoGou Data	Nutch	W1-6
	Sort	Offline Analytics	Wikipedia Entries	MPI, Spark, Hadoop	W1-7
	Read	Cloud OLTP	ProfSearch Resumes	HBase, Mysql	W1-11-1
	Write	Cloud OLTP	ProfSearch Resumes	HBase, Mysql	W1-11-2
	Scan	Cloud OLTP	ProfSearch Resumes	HBase, Mysql	W1-11-3
Social Network	CC	Offline Analytics	Facebook Social Network	MPI, Spark, Hadoop	W2-8-1
	Kmeans	Offline Analytics	Facebook Social Network	MPI, Spark, Hadoop	W2-8-2
	BFS	Offline Analytics	Self Generating by the program	MPI	W2-9
E-commerce	Select Query	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-1
	Aggregation Query	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-2
	Join Query	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-3
	CF	Offline Analytics	Amazon Movie Review	hadoop, Spark, MPI	W3-4
	Bayes	Offline Analytics	Amazon Movie Review	hadoop, Spark, MPI	W3-5
	Project	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-1
	Filter	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-2
	Cross Product	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-3
	OrderBy	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-4
	Union	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-5
	Difference	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-6
	Aggregation	Interactive Analytics	E-commerce Transaction Data	Hive, Shark, Impala	W3-6-7
Multimedia	BasicMPEG	Offline Analytics	stream data	Libc	W4-1
	SIFT	Offline Analytics	ImageNet	MPI	W4-2-1
	DBN	Offline Analytics	MNIST	MPI	W4-2-2
	Speech Recognition	Offline Analytics	audio files	MPI	W4-3
	Ray Tracing	Offline Analytics	scene description files	MPI	W4-4
	Image Segmentation	Offline Analytics	ImageNet	MPI	W4-5
	Face Detection	Offline Analytics	ImageNet	MPI	W4-6
Bio-informatics	SAND	Offline Analytics	Genome sequence data	Work Queue	W5-1
	BLAST	Offline Analytics	Assembly of the human genome	MPI	W5-2

[1]The workload ID of BigDataBench 3.1 corresponds with the workload ID in the BigDataBench specification which can be found from *BigDataBench Specification and Implementation*

Table 3. The summary of different Big Data Benchmarks.

	Specification	Application domains	Workload types	Workloads	Scalable data sets abstracting from real data	Multiple implementations	Multi-tenancy	Subsets	Simulator version
BigData Bench	Y	five	four[1]	thirty-three[2]	eight[3]	Y	Y	Y	Y
BigBench	Y	one	three	ten	three	N	N	N	N
CloudSuite	N	N/A	two	eight	three	N	N	N	Y
HiBench	N	N/A	two	ten	three	N	N	N	N
CALDA	Y	N/A	one	five	N/A	Y	N	N	N
YCSB	Y	N/A	one	six	N/A	Y	N	N	N
LinkBench	Y	N/A	one	ten	one	Y	N	N	N
AMP Benchmarks	Y	N/A	one	four	N/A	Y	N	N	N

[1]The four workload types are Offline Analytics, Cloud OLTP, Interactive Analytics and Online Service
[2]There are 42 workloads in the specification. We have implemented 33 workloads
[3]There are 8 real data sets can be scalable, other 6 ones are ongoing development

2.3 What is new in BigDataBench 3.1

We updated the Benchmarking methodology and added two new application domains: Multimedia and Bioinformatics. Now, there are five typical application domains: Search Engine, Social Network, E-commerce, Multimedia and Bioinformatics in BigDataBench 3.1. With the new methodology, we proposed the Benchmark specification for each application domain, and defined data sets and workloads in the application domains. Based on the specification, we implemented the BigDataBench 3.1. Now it includes 14 real-world data sets, and 33 big data workloads. The Multi tenancy version for Cloud computing communities and simulator version for architecture communities are also released.

3 Big Data Benchmarking Methodology

Figure 2 summarizes the benchmarking methodology in BigDataBench. Overall, it involves five steps: investigating and choosing important application domains; identifying typical workloads and data sets; proposing big data benchmark specifications; providing diverse implementations using competitive techniques; mixing different workloads to assemble multi-tenancy workloads or subsetting big data benchmarks.

First of all, we investigated the dominant application domains of internet services according to widely acceptable metrics—the number of page views and

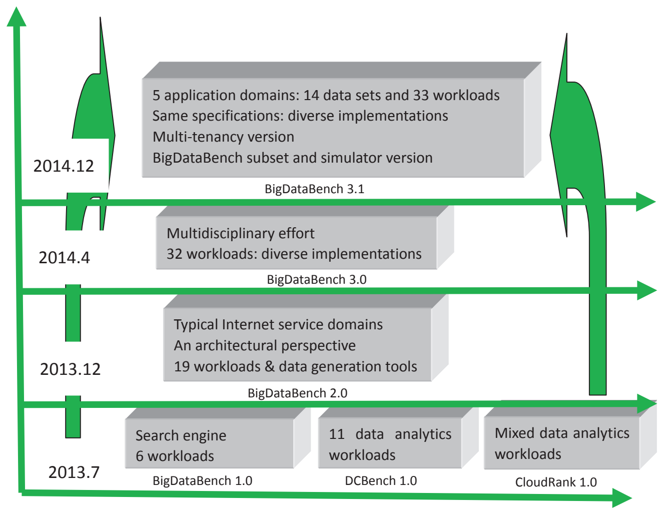
5 application domains: 14 data sets and 33 workloads
Same specifications: diverse implementations
Multi-tenancy version
BigDataBench subset and simulator version
2014.12
BigDataBench 3.1
Multidisciplinary effort
32 workloads: diverse implementations
2014.4
BigDataBench 3.0
Typical Internet service domains
An architectural perspective
19 workloads & data generation tools
2013.12
BigDataBench 2.0
Search engine
6 workloads
11 data analytics workloads
Mixed data analytics workloads
2013.7
BigDataBench 1.0
DCBench 1.0
CloudRank 1.0

Fig. 1. BigDataBench Evolution

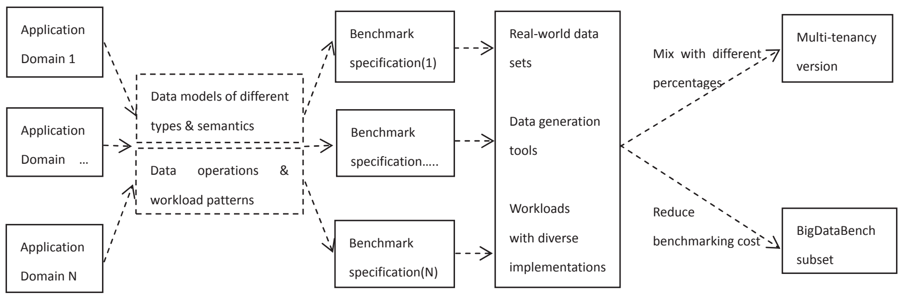
Application Domain 1
Application Domain ...
Application Domain N
Data models of different types & semantics
Data operations & workload patterns
Benchmark specification(1)
Benchmark specification.....
Benchmark specification(N)
Real-world data sets
Data generation tools
Workloads with diverse implementations
Mix with different percentages
Reduce benchmarking cost
Multi-tenancy version
BigDataBench subset

Fig. 2. BigDataBench benchmarking methodology.

daily visitors. According to the analysis in [6], the top three application domains are search engines, social networks, and e-commerce, taking up 80% page views of all the internet services in total. Meanwhile, multimedia data analytics and bioinformatics are two emerging but important big data application domains. So we selected out those five important applications domains: search engine, social network, e-commerce, multimedia data analytics and bioinformatics. At the second step, we analyzed typical workloads and data sets in each domain from two perspectives: diverse data models, i.e., structured, semi-structured, and unstructured, and different semantics, e.g., text, graph, multimedia data; identifying frequent-appearing data operations and workload patterns. After that, we proposed big data benchmarks specifications for each domain. At the fourth step, we implemented the same specifications using competitive techniques. For example, for offline analytics workloads, we implemented the workloads using

MapReduce, MPI, Spark and DataMPI. Meanwhile, we choosed real-world data sets, and then provides parallel big data generation tools to generate scalable big data while preserving their original characteristics. Finally, we provided the multi-tenancy version benchmark and the subset benchmark suites for different purposes. We provide the multi-tenancy version of BigDataBench, which allows flexible setting and replaying of mixed workloads with different percentages. To reduce the research or benchmarking cost, we select a small number of representative benchmarks, which we call the BigDataBench subset, from a large amount of BigDataBench workloads according to workload characteristics from a specific perspective.

4 BigDataBench Subsetting

4.1 Motivation

For system and architecture researches (i. e., architecture, OS, networking and storage), the number of benchmarks will be multiplied by different implementations, and hence becoming massive. For example, BigDataBench 3.1 provides about 77 workloads (with different implementations). Given the fact that it is expensive to run all the workloads, especially for architectural researches, which usually evaluate new designs using simulators, downsizing the full range of the BigDataBench 3.1 benchmark suite to a subset of necessary (non-substitutable) workloads is essential to guarantee cost-effective benchmarking and simulations.

4.2 Methodology

A brief description of our subsetting methodology is listed below. The details of subsetting (downsizing) workloads are summarized in [24].

1. Identify a comprehensive set of workload characteristics from a specific perspective, which affects the performance of workloads.
2. Eliminate the correlation data in those metrics and map the high dimension metrics to a low dimension.
3. Use the clustering method to classify the original workloads into several categories and choose representative workloads from each category.

4.3 Architecture subset

Microarchitectural Metric Selection We select a broad set of metrics of different types that cover all major characteristics. We particularly focus on factors that may affect data movement or calculation. For example, a cache miss may delay data movement, and a branch misprediction flushes the pipeline.

Table 4 summarizes the 45 metrics we used and we categorize them below.

Instruction Mix. The instruction mix can reflect a workload's logic and affect performance. Here we consider both the instruction type and the execution mode (i.e., user mode running in ring three and kernel mode running in ring zero).

Cache Behavior. The processor in our experiments has private L1 and L2 caches per core, and all cores share an L3. The L1 cache is shared for instructions and data. The L2 and L3 are unified. We track the cache misses per kilo instructions and cache hits per kilo instructions except L1 data cache, note that for the L1D miss penalties may be hidden by out-of-order cores.

Translation Look-aside Buffer (TLB) Behavior. Modern processors have multiple levels of TLB (most of them are two-level). The first level has separate instruction and data TLBs. The second level is shared. We collect statistics at both levels.

Branch Execution. We consider the miss prediction ratio and the ratio of branch instructions executed to those retired. These reflect how many branch instructions are predicted wrong and how many are flushed.

Pipeline Behavior. Stalls can happen in any part of the pipeline, but superscalar out-of-order processors prevent us from precisely breaking down the execution time [26, 23]. Retirement-centric analysis also has difficulty accounting for how the CPU cycles are used because the pipeline continues executing instructions even when retirement is blocked [27]. Here we focus on counting cycles stalled due to resource conflicts, e.g., reorder buffer full stalls that prevent new instructions from entering the pipeline.

Offcore Requests and Snoop Responses. Offcore requests tell us about individual core requests to the LLC (Last Level Cache). Requests can be classified into data requests, code requests, data write-back requests, and request for ownership (RFO) requests. Snoop responses give us information on the workings of the cache coherence protocol.

Parallelism. We consider Instruction Level Parallelism (ILP) and Memory Level Parallelism (MLP). ILP reflects how many instructions can be executed in one cycle (i.e., the IPC), and MLP reflects how many outstanding cache requests are being processed concurrently.

Operation Intensity. The ratio of computation to memory accesses reflects a workload's computation pattern. For instance, most big data workloads have a low ratio of floating point operations to memory accesses, whereas HPC workloads generally have high floating point operations to memory accesses ratios [30].

Removing Correlated Data The BigDataBench 3.1 includes 77 workloads. Given the 77 workloads and 45 metrics for each workload, it is difficult to analyze all the metrics to draw meaningful conclusions. Note, however, that some metrics may be correlated. For instance, long latency cache misses may cause pipeline stalls. Correlated data can skew analysis — many correlated metrics will overemphasize a particular property's importance. So we eliminate correlated data before analysis. Principle Component Analysis (PCA) [25] is a common method for removing such correlated data [29, 21, 22, 19]. We first normalize metric values to a Gaussian distribution with mean equal to zero and standard deviation equal to one (to isolate the effects of the varying ranges of each dimension). Then we use Kaiser's Criterion to choose the number of principle components (PCs). That is, only the top few PCs, which have eigenvalues greater than or equal to one, are

Table 4. Microarchitecture Level Metrics.

Category	No.	Metric Name	Description
Instruction Mix	1	LOAD	load operations' percentage
	2	STORE	store operations' percentage
	3	BRANCH	branch operations' percentage
	4	INTEGER	integer operations' percentage
	5	FP	X87 floating point operations' percentage
	6	SSE FP	SSE floating point operations' percentage
	7	KERNEL MODE	the ratio of instruction running on kernel mode
	8	USER MODE	the ratio of instruction running on user mode
	9	UOPS TO INS	the ratio of micro operation to instruction
Cache Behavior	10	L1I MISS	L1 instruction cache misses per K instructions
	11	L1I HIT	L1 instruction cache hits per K instructions
	12	L2 MISS	L2 cache misses per K instructions
	13	L2 HIT	L2 cache hits per K instructions
	14	L3 MISS	L3 cache misses per K instructions
	15	L3 HIT	L3 cache hits per K instructions
	16	LOAD HIT LFB	loads miss the L1D and hit line fill buffer per K instructions
	17	LOAD HIT L2	loads hit L2 cache per K instructions
	18	LOAD HIT SIBE	loads hit sibling core's L2 cache per K instructions
	19	LOAD HIT L3	loads hit unshared lines in L3 cache per K instructions
	20	LOAD LLC MISS	loads miss the L3 cache per K instructions
TLB Behavior	21	ITLB MISS	misses in all levels of the instruction TLB per K instructions
	22	ITLB CYCLE	the ratio of instruction TLB miss page walk cycles to total cycles
	23	DTLB MISS	misses in all levels of the data TLB per K instructions
	24	DTLB CYCLE	data TLB miss page walk cycles to total cycles
	25	DATA HIT STLB	DTLB first level misses that hit in the second level TLB per K instructions
Branch Execution	26	BR MISS	branch miss prediction ratio
	27	BR EXE TO RE	the ratio of executed branch instruction to retired branch execution
Pipeline Behavior	28	FETCH STALL	the ratio of instruction fetch stalled cycle to total cycles
	29	ILD STALL	the ratio of Instruction Length Decoder stalled cycle to total cycles
	30	DECODER STALL	the ratio of Decoder stalled cycles to total cycles
	31	RAT STALL	the ratio of Register Allocation Table stalled cycles to total cycles
	32	RESOURCE STALL	the ratio of resource related stalled to total cycles, which including load store buffer full stalls, Reservation Station full stalls, ReOrder buffer full stalls and etc
	33	UOPS EXE CYCLE	the ratio of micro operation executed cycle to total cycles
	34	UOPS STALL	the ratio of no micro operation executed cycle to total cycles
Offcore Request	35	OFFCORE DATA	percentage of offcore data request
	36	OFFCORE CODE	percentage of offcore code request
	37	OFFCORE RFO	percentage of offcore Request For Ownership
	38	OFFCORE WB	percentage of data write back to uncore
Snoop Response	39	SNOOP HIT	HIT snoop responses per K instructions
	40	SNOOP HITE	HIT Exclusive snoop responses per K instructions
	41	SNOOP HITM	HIT Modified snoop responses per K instructions
Parallelism	42	ILP	Instruction Level Parallelism
	43	MLP	Memory Level Parallelism
Operation Intensity	44	INT TO MEM	integer computation to memory access ratio
	45	FP TO MEM	floating point computation to memory access ratio

kept. With Kaiser's Criterion, the resulting data is ensured to be uncorrelated while capturing most of the original information. Finally we choose nine PCs, which retain 89.3% variance.

Clustering We use K-Means clustering on the nine principle components obtained from the PCA algorithm to group workloads into similarly behaving application clusters and then we choose one representative workload from each cluster. In order to cluster all the workloads into reasonable classes, we use the Bayesian Information Criterion (BIC) to choose the proper K value. The BIC is a measure of the "goodness of fit" of a clustering for a data set. The larger the BIC scores, the higher the probability that the clustering is a good fit to the data. Here we determine the K value that yields the highest BIC score.

We use the formulation from Pelleg et al. [28] shown in Equation 1 to calculate the BIC.

$$BIC(D, K) = l(D|K) - \frac{p_j}{2} log(R) \tag{1}$$

Where D is the original data set to be clustered. In this Section, D is 77×9 matrix which indicates 77 workloads and each workload is represented by 9 PCs (Principle Components). $l(D|K)$ is the likelihood. R is the number of workloads to be clustered. p_j is the sum of $K-1$ class probabilities, which is $K + dK$. d is the dimension of each workloads, which is $K + dK$, which is 9 for we choose 9 PCs. To compute $l(D|K)$, we use Equation 2.

$$l(D|K) = \sum_{i=1}^{K} (-\frac{R_i}{2} log(2\pi) - \frac{R_i \cdot d}{2} log(\sigma^2) - \frac{R_i - K}{2} + R_i log R_i - R_i log R) \tag{2}$$

Where R_i is the number of points in the i^{th} cluster, and σ^2 is the average variance of the Euclidean distance from each point to its cluster center, which is calculate by Equation 3.

$$\sigma^2 = \frac{1}{R-K} \sum_i (x_i - \mu(i))^2 \tag{3}$$

Here x_i is the data point assigned to cluster i, and $\mu(i)$ represents the center coordinates of cluster i.

We ultimately cluster the 77 workloads (all big data workloads in BigDataBench 3.0) into 17 groups, which are listed in Table 5.

Representative Workloads Selection There are two methods to choose the representative workload from each cluster. The first is to choose the workload that is as close as possible to the center of the cluster it belongs to. The second one is to select an extreme workload situated at the "boundary" of each cluster.

Combined with hierarchical clustering result, we select the workload situated at the "boundary" of each cluster as the architecture subset of BigDataBench 3.1.

The rationale behind the approach would be that the behavior of the workloads in the middle of a cluster can be extracted from the behavior of the boundary, for example through interpolation. So the representative workloads are listed in Table 6. And the number of workloads that each selected workload represents is given in the third column.

In the case that researchers need the workloads which are chosen by the first method (i.e., choosing the workload that is as close as possible to the center of the cluster), we also list them in Table 7.

5 BigDataBench Simulator Version

We deploy the architecture subset on MARSSx86 [11] and Simics [14] respectively and release them as BigDataBench simulator version. This section gives a brief introduction of these two computer architecture simulators and how to run the workloads in each of them.

5.1 Motivation

A full system architecture simulator effectively provides virtual hardware that is independent of the nature of the host computer. The full-system model typically has to include processor cores, peripheral devices, memories, interconnection buses, and network connections. Architecture simulators, which aim at allowing accurate timings of the processor, are very useful in the following aspects:

- Obtaining detailed performance characteristics: A single execution of simulators can generate a large set of performance data, which can be analyzed offline.
- Evaluating different hardware designs without building expensive physical hardware systems.
- Debugging on simulator to detect the potential errors instead of on real hardware, which requires re-booting and re-running the code to reproduce the problems.

We provide the BigDataBench simulator version to facilitate the big data researches in the above aspects. Simulation is a time consuming activity. It is prohibitively expensive to run all big data application in BigDataBench-v3.1. So we just deploy the architecture subset application mentioned in Section 4.3, i.e. the application in Table 6, on those two simulators and release the image as BigDataBench simulator version.

5.2 MARSSx86 Version

MARSSx86 is an open source, fast, full system simulation tool built on Qemu to support cycle-accurate simulation of superscalar homogeneous and heterogeneous multicore x86 processors [11]. MARSSx86 includes detailed models of coherent caches, interconnections, chipsets, memory and IO devices. MARSSx86 can simulate the execution of all software components in the system, including unmodified binaries of applications, operating systems and libraries.

BigDataBench MARSSx86 version overview The MARSSx86 has the following characteristics:

- Good performance and accuracy: average simulated commit rate of 200K+ instructions/second.
- Qemu based full system emulation environment with models for chipset and peripheral devices.
- Detailed models for coherent caches and on-chip interconnections.

MARSSx86 user Guide
System Requirements
MARSS runs a Linux platform with the following minimum requirements:

- x86_64 CPU cores with a minimum 2GHz clock and 2GB RAM (4GB RAM is preferred).
- C/C++ compiler, gcc or icc; SCons compilation tool minimum version 1.2.
- SDL Development Libraries (required for QEMU).

Deploying MARSS and Running Big Data Applications
Once meeting the above pre-requirements, compiling MARSS is simple. What users need to do is as follows:

1. Download the appropriate MARSS installation package from the web site.
2. Extract the installation package:

```
tar xf marss-0.4.tar.gz
```

3. Enter the temporary installation directory, and run the command as follows:

```
$ cd marss-0.4
$ scons -Q
```

4. By default it will compile the MARSS for single simulated core. To simulate more than one core, e.g., SMP or CMP configuration, users should add an option 'c=NUM_CORES' to compile MARSS as shown below. This command will compile the MARSS to simulate 8 cores:

```
$ scons -Q c=8
```

5. We provide four qemu-disk-images and two qemu-network-config-scripts:
   - marss-1.img (the qemu-disk-image of master node to run Impala based workloads)
   - marss-2.img (the qemu-disk-image of slaver node to run Impala based workloads)
   - marss-3.img (the qemu-disk-image of master node to run Hadoop and Spark based workloads)
   - marss-4.img (the qemu-disk-image of slaver node to run Hadoop and Spark based workloads)
   - qemu-ifup (qemu-network-config-script for master node)
   - qemu-ifup2 (qemu-network-config-script for slaver node, you should run this script before qemu-ifup)

   To run Impala based workloads of BigDataBench, you should use following commands to run MARSS:

```
master: $ qemu/qemu-system-x86_64 -m 8192 -hda [path-to-marss-1.img] -monitor stdio -net nic,macaddr=52:54:00:12:34:55 -net tap,ifname=tap1,script=[path-to-qemu-ifup2]
```

```
slaver: $ qemu/qemu-system-x86_64 -m 8192 -hda [path-to-marss-2.img] -monitor stdio -net nic -net tap,ifname=tap0,script=[path-to-qemu-ifup]
```

   To run Hadoop, Hive, Spark, or Shark based workloads of BigDataBench, you should use following commands to run MARSS:

```
master: $ qemu/qemu-system-x86_64 -m 8192 -hda [path-to-marss-3.img] -monitor stdio -net nic,macaddr=52:54:00:12:34:55 -net tap,ifname=tap1,script=[path-to-qemu-ifup2]
```

```
slaver: $ qemu/qemu-system-x86_64 -m 8192 -hda [path-to-marss-4.img] -monitor stdio -net nic -net tap,ifname=tap0,script=[path-to-qemu-ifup]
```

6. You can use all of the regular Qemu commands. Once the VM is booted, the host's command line has become the VM console and you can start the benchmark application, issue following commands in that console:

```
(qemu) simconfig -run -stopinsns 100m -stats [stats-filename] -machine
MACHINE_NAME
```

   You can find the MACHINE_NAME and hardware configuration in the marss-0.4/config path. The MACHINE_NAME should be "shared_l2" or "private_l2" if you follow the commands above.

The above paragraphs shows how to run Impala based workloads. Users can use different queries by modifying the *runMicroBenchmark.sh*. For other workloads users can boot the MARSS and use the commands in Section 9.

5.3 Simics 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.

BigDataBench Simics version overview We use SPARC as the instruction set architecture in our Simics version simulator benchmark suite, and deploy Solaris operation systems, since the X86 architecture is not well supported by some simulators components based on Simics. For instance the Flexus [13], which is a family of component-based C++ computer architecture simulators that build on Simics Micro-Architecture Interface, does not support our-of-order mode for x86 architecture.

Simics user Guide Simics is recommended to install in the /opt/virtutech directory by using the following commands.

1. Download the appropriate Simics installation package from the website, such as simics-pkg-00-3.0.0-linux.tar.
2. Extract the installation package:

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

3. Enter the temporary installation directory and run the install script using the command as follows:

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

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

5. When the installation script is finished, Simics has been installed in the /opt/virtutech/simics-<version>.
6. When the Simics is successfully installed, the temporary installation directory can be deleted.

The detailed commands of how to run big data workloads in Simics can be found in Section 9.

6 Multi-tenancy of BigDataBench

6.1 Background of Multi-tenancy

What is Multi-tenancy Datacenters? Data center reflects the thinking that the network is the computer, which makes the amount of computing resource, storage resources and software resources linked together, then forming a huge shared virtual IT resources pool to provide services via the Internet. Data center focuses on the high concurrency, the diversity of application performance, low power, automation, high efficiency.

Within this context, a multi-tenant datacenter can be explained from three perspectives:

– Resource pooling and broad network access. Infrastructure resources such as VM, storage, and networking are pooled and shared among multiple cloud consumers.
– On-demand and elastic resource provision. Cloud consumers can get any quantity of resources at any time according to their demand.
– Metered resources. Resources are charged in a pay-as-you-go manner like electricity and water.

Existing problems Existing big data benchmarks typically focus on latency/throughput for a single run of workload performed in a dedicated set of machines. The benchmarking process is too synthetic that it does not match the typically operating conditions of real systems, where *mixes of different percentages of tenants and workloads share the same computing infrastructure.* For such an issue, benchmark suite that support real-world scenarios serving tenants with different amounts of users and heterogeneous workloads are urgently needed.

How to characterize datacenter tenants? Datacenter tenants can be characterized from three aspects:

– The number of tenants (scalability of benchmark): Does the system scale well with the number of tenants? How many tenants are able to run in parallel?
– The priorities of tenants (Fairness of benchmark): How fair is the system, i.e., are the available resources equally available to all tenants? If tenants have different priorities?
– Time line: how the number and priorities of tenants change over time?

How to characterize big data workloads? Big data workloads can be characterized from three aspects:

– Data properties, including data types and sources, and input/output data volumes, distributions.
– Computation semantics, including source codes (implementation logic of workloads) and the software stacks.
– Job arrival patterns, including requests' arrival rate and sequences.

6.2 Definition of Multi-tenancy version

Multi-tenancy version of BigDataBench is a benchmark suite aiming at supporting the scenarios of multiple tenants running heterogeneous applications in data centers. Examples are latency-critical online services (e.g., web search engine) and latency-insensitive offline batch applications. The basic idea of Multi-tenancy version is to understand the behavior of realistic big data workloads (involving both service and batch application workloads) and their users. The specification of Multi-tenancy version is shown in Figure 3. The workload suite has been designed and implemented based on workload traces from real-world applications, allowing the flexible setting and replaying of these workloads according to users' varying requirements. At present, multi-tenancy version consists of two types of representative workloads: Nutch search engine and Hadoop MapReduce workloads, which correspond to three real-world workload traces: Sougou, Facebook trace, and Google trace, respectively.

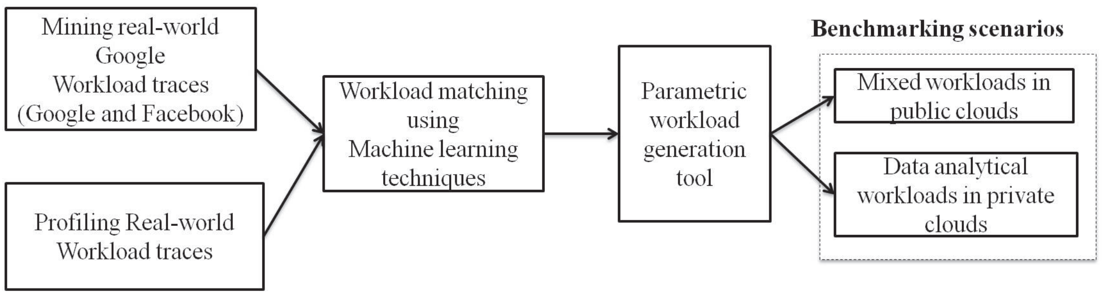
Mining real-world Google Workload traces (Google and Facebook)
Profiling Real-world Workload traces
Workload matching using Machine learning techniques
Parametric workload generation tool
Benchmarking scenarios
Mixed workloads in public clouds
Data analytical workloads in private clouds

Fig. 3. Overview of Multi-tenancy version of BigDataBench.

Main Features. Multi-tenancy version is currently integrated with Hadoop and Nutch Search Engine. We believe that data center cluster operators can use multi-tenancy version to accomplish other previously challenging tasks, including but not limited to resource provisioning and planning in multiple dimensions; configurations tuning for diverse job types within a workload; anticipating workload consolidation behavior and quantify workload superposition in multiple dimensions.

The multi-tenancy version has the following five features:

– Repository of workload traces and real life Search-engine workloads from production systems.
– Applying robust machine learning algorithm to match the workload characteristics information from both real workloads and workload traces, thus exacting basis for workload replaying.
– Workload synthesis tools to generate representative test workloads by parsing workload replaying basis.
– Convenient multi-tenancy workload replay tools to execute both time-critical and analytical workloads with low performance overhead.
– Scenarios of both *mixed workloads in public clouds* and *data analytical workloads in private clouds.*

References

1. http://www.tingvoa.com.
2. ftp://ftp.tek.com/tv/test/streams/Element/index.html/.
3. http://jedi.ks.uiuc.edu/ johns/raytracer/.
4. http://ccl.cse.nd.edu/software/sand/.
5. http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/.
6. Alexa topsites. http://www.alexa.com/topsites/global;0.
7. Amazon movie reviews. http://snap.stanford.edu/data/web-Amazon.html.
8. Daiad. http://www.daiad.eu/wp-content/uploads/2014/10/D1.2_DAIAD_Requirements_and_Architecture_v1.0
9. Facebook graph. http://snap.stanford.edu/data/egonets-Facebook.html.
10. Google web graph. http://snap.stanford.edu/data/web-Google.html.
11. Micro-architectural and system simulator for x86-based systems (MARSSx86) website. http://marss86.org/ marss86/index.php/Home.
12. mnist. http://yann.lecun.com/exdb/mnist/.
13. Simflex fast, accurate & flexible computer architecture simulation. http://parsa.epfl.ch/simflex/.
14. Simics website. http://www.windriver.com/simics/.
15. Sogou labs. http://www.sogou.com/labs/.
16. Standard performance evaluation corporation (spec) website. http://www.spec.org.
17. wikipedia. http://en.wikipedia.org.
18. C. Bienia. *Benchmarking Modern Multiprocessors.* PhD thesis, Princeton University, January 2011.
19. C. Bienia and K. Li. Fidelity and scaling of the PARSEC benchmark inputs. In *IEEE International Symposium on Workload Characterization*, pages 1–10, Dec. 2010.
20. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

21. L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Workload design: Selecting representative program-input pairs. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 83–94, Sep. 2002.
22. L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Quantifying the impact of input data sets on program behavior and its applications. *Journal of Instruction-Level Parallelism*, 5(1):1–33, 2003.
23. S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith. A performance counter architecture for computing accurate CPI components. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 175–184, Oct. 2006.
24. Z. Jia, L. Wang, J. Zhan, L. Zhang, and C. Luo. Characterizing data analysis workloads in data centers. In *Workload Characterization (IISWC), 2013 IEEE International Symposium on*. IEEE.
25. I. Jolliffe. *Principal Component Analysis*. Wiley Online Library, 2005.
26. K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker. Performance characterization of a Quad Pentium Pro SMP using OLTP workloads. In *International Symposium on Computer Architecture*, Jun. 1998.
27. D. Levinthal. Cycle accounting analysis on Intel Core 2 processors. https://software.intel.com/sites/products/collateral/hpc/vtune/cycle_accounting_analysis.pdf, cited Apr. 2014.
28. D. Pelleg and A. W. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *International Conference on Machine Learning*, pages 727–734, Jun. 2000.
29. A. Phansalkar, A. Joshi, and L. John. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In *International Symposium on Computer Architecture*, Jun. 2007.
30. L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, et al. Bigdatabench: A big data benchmark suite from internet services. *The 20th IEEE International Symposium On High Performance Computer Architecture (HPCA)*, 2014.
31. Y. Zhu, J. Zhan, C. Weng, R. Nambiar, J. Zhang, X. Chen, and L. Wang. BigOP: Generating comprehensive big data workloads as a benchmarking framework. In *Database Systems for Advanced Applications*, pages 483–492. Springer, 2014.

Table 5. Clustering Results

# Cluster	Workloads
1	Cloud-OLTP-Read, Impala-JoinQuery, Shark-Difference, Hadoop-Sort, Cloud-OLTP-San, Ipala-TPC-DS-query8, Impala-Crossproduct, Impala-Project, Impala-AggregationQuery, Cloud-OLTP-Write
2	Hive-TPC-DS-query10, Hive-TPC-DS-query12-1, Hive-Difference, Hadoop-Index, Hive-TPC-DS-query6, Hive-TPC-DS-query7, Hive-TPC-DS-query9, Hive-TPC-DS-query13, Hive-TPC-DS-query12-2
3	Hive-Orderby, Hive-SelectQuery, Hive-TPC-DS-query8, Impala-SelectQuery, Hive-Crossproduct, Hive-Project, Hive-JoinQuery, Hive-AggregationQuery
4	Impala-TPC-DS-query6, Impala-TPC-DS-query12_2, Hive-TPC-DS-query3,Spark-NaiveBayes, Impala-TPC-DS-query7, Impala-TPC-DS-query13, Impala-TPC-DS-query9, Impala-TPC-DS-query10, Impala-TPC-DS-query3
5	Shark-Union, Spark-WordCount, Shark-Aggregation-AVG, Shark-Filter, Shark-Aggregation-MAX, Shark-SelectQuery, Shark-Aggregation-MIN, Shark-Aggregation-SUM
6	Impala-Filter, Impala-Aggregation-AVG, Impala-Union, Impala-Orderby, Impala-Aggregation-MAX, Impala-Aggregation-MIN, Impala-Aggregation-SUM
7	Hive-Aggregation-AVG, Hive-Aggregation-MIM, Hive-AggregationSUM, Hadoop-Grep, Hive-Union, Hive-AggregationMAX, Hive-Filter, Hadoop-Pagerank
8	Shark-TPC-DS-query9, Shark-TPC-DS-query7, Shark-TPC-DS-query10, Shark-TPC-DS-query3
9	Shark-AggregationQuery, Shark-TPC-DS-query6, Shark-Project, Shark-TPC-DS-query13
10	Shark-JoinQuery, Shark-Orderby, Shark-Crossproduct
11	Spark-Kmeans
12	Shark-TPCDS-query8
13	Spark-Pagerank
14	Spark-Grep
15	Hadoop-WordCount
16	Hadoop-NaiveBayes
17	Spark-Sort

Table 6. Treat the marginal ones as representative workloads

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

Table 7. Treat the central ones as representative workloads

No.	Workload name	Number of workloads in its cluster
1	Cloud-OLTP-Write	10
2	Hive-TPC-DS-query13	9
3	Hive-AggregationQuery	9
4	Impala-TPC-DS-query6	9
5	Shark-Union	8
6	Impala-Aggregation-MAX	7
7	Hive-Aggregation-AVG	7
8	Shark-TPC-DS-query7	4
9	Shark-TPC-DS-query6	4
10	Shark-Crossproduct	3
11	Spark-Kmeans	1
12	Shark-TPC-DS-query8	1
13	Spark-Pagerank	1
14	Spark-Grep	1
15	Hadoop-WordCount	1
16	Hadoop-NaiveBayes	1
17	Spark-Sort	1