BigDataBench: a Big Data Benchmark Suite from Internet Services

Lei Wang[1,7], Jianfeng Zhan *[1], Chunjie Luo[1], Yuqing Zhu[1], Qiang Yang[1], Yongqiang He[2], Wanling Gao[1], Zhen Jia[1], Yingjie Shi[1], Shujie Zhang[3], Chen Zheng[1], Gang Lu[1], Kent Zhan[4], Xiaona Li[5], and Bizhu Qiu[6]

[1]State Key Laboratory of Computer Architecture (Institute of Computing Technology, Chinese Academy of Sciences)
{wanglei_2011, zhanjianfeng, luochunjie, zhuyuqing, yangqiang, gaowanling, jiazhen, shiyingjie, zhengchen, lugang}@ict.ac.cn
[2]Dropbox, yq@dropbox.com
[3]Huawei, shujie.zhang@huawei.com
[4]Tencent, kentzhan@tencent.com
[5]Baidu, lixiaona@baidu.com
[6]Yahoo!, qiubz@yahoo-inc.com
[7]University of Chinese Academy of Sciences, China

Abstract

As architecture, systems, and data management communities pay greater attention to innovative big data systems and architecture, the pressure of benchmarking and evaluating these systems rises. However, the complexity, diversity, frequently changed workloads, and rapid evolution of big data systems raise great challenges in big data benchmarking. Considering the broad use of big data systems, for the sake of fairness, big data benchmarks must include diversity of data and workloads, which is the prerequisite for evaluating big data systems and architecture. Most of the state-of-the-art big data benchmarking efforts target evaluating specific types of applications or system software stacks, and hence they are not qualified for serving the purposes mentioned above.

*This paper presents our joint research efforts on this issue with several industrial partners. Our big data benchmark suite—*BigDataBench *not only covers broad application scenarios, but also includes diverse and representative data sets. Currently, we choose 19 big data benchmarks from dimensions of application scenarios, operations/ algorithms, data types, data sources, software stacks, and application types, and they are comprehensive for fairly measuring and evaluating big data systems and architecture.* BigDataBench *is publicly available from the project home page http://prof.ict.ac.cn/BigDataBench.*

Also, we comprehensively characterize 19 big data workloads included in BigDataBench *with varying data inputs. On a typical state-of-practice processor, Intel Xeon E5645, we have the following observations: First, in comparison with the traditional benchmarks: including* PARSEC, HPCC, *and* SPECCPU, *big data applications have very low* operation intensity, *which measures the ratio of the total number of instructions divided by the total byte number of memory accesses; Second, the volume of data input has non-negligible impact on micro-architecture characteristics, which may impose challenges for simulation-based big data architecture research; Last but not least, corroborating the observations in* CloudSuite *and* DCBench *(which use smaller data inputs), we find that the numbers of L1 instruction cache (L1I)* misses per 1000 instructions *(in short, MPKI) of the big data applications are higher than in the traditional benchmarks; also, we find that L3 caches are effective for the big data applications, corroborating the observation in* DCBench.

1 Introduction

Data explosion is an inevitable trend as the world is connected more than ever. Data are generated faster than ever, and to date about 2.5 quintillion bytes of data are created daily [1]. This speed of data generation will continue in the coming years and is expected to increase at an exponential level, according to IDC's recent survey. The above fact gives birth to the widely circulated concept *Big Data*. But turning big data into insights or true treasure demands an in-depth extraction of their values, which heavily relies upon and hence boosts deployments of massive big data systems. As architecture, systems, and data management communities pay greater attention to innovative big data systems and architecture [13, 17], [31], the pressure of measuring, comparing, and evaluating these systems rises [19]. Big data benchmarks are the foundation of those ef-

*The corresponding author is Jianfeng Zhan.

forts [18]. However, the complexity, diversity, frequently changed workloads—so called workload churns [13], and rapid evolution of big data systems impose great challenges to big data benchmarking.

First, there are many classes of big data applications without comprehensive characterization. Even for internet service workloads, there are several important application domains, e.g., search engines, social networks, and e-commerce. Meanwhile, the value of big data drives the emergence of innovative application domains. The diversity of data and workloads needs comprehensive and continuous efforts on big data benchmarking. Second, most big data applications are built on the basis of complex system software stacks, e.g., widely used Hadoop systems. However, there are not one-size-fits-all solutions [27], and hence big data system software stacks cover a broad spectrum. Third, even if some big data applications are mature in terms of business and technology, customers, vendors, or researchers from academia or even different industry domains do not know enough about each other. The reason is that most internet service providers treat data, applications, and web access logs as business confidential, which prevents us from building benchmarks.

As summarized in Table 1, most of the state-of-the-art big data benchmark efforts target evaluating specific types of applications or system software stacks, and hence fail to cover diversity of workloads and real-world data sets. However, considering the broad use of big data systems, *for the sake of fairness, big data benchmarks must include diversity of workloads and data sets, which is the prerequisite for evaluating big data systems and architecture*. This paper presents our joint research efforts on big data benchmarking with several industrial partners. Our methodology is from real systems, covering not only broad application scenarios but also diverse and representative real-world data sets. Since there are many emerging big data applications, we take an incremental and iterative approach in stead of a top-down approach. After investigating typical application domains of internet services—an important class of big data applications, we pay attention to investigating workloads in three most important application domains according to widely acceptable metrics—the number of page views and daily visitors, including search engine, e-commerce, and social network. To consider workload candidates, we make a tradeoff between choosing different types of applications: including online services, offline analytics, and realtime analytics. In addition to workloads in three main application domains, we include micro benchmarks for different data sources, "Cloud OLTP" workloads[1], and relational queries workloads, since they are fundamental and widely used. For three types of big data applications, we include both widely-used and state-of-the-art system software stacks.

From search engines, social networks, and e-commerce domains, six representative real-world data sets, whose varieties are reflected in two dimensions of data types and data sources, are collected, with the whole spectrum of data types including structured, semi-structured, and unstructured data. Currently, the included data sources are text, graph, and table data. Using these real data sets as the seed, the data generators [23] of *BigDataBench* generate synthetic data by scaling the seed data while keeping the data characteristics of raw data. To date, we chose and developed nineteen big data benchmarks from dimensions of application scenarios, operations/ algorithms, data types, data sources, software stacks, and application types. We also plan to provide different implementations using the other software stacks. All the software code is available from [6].

On a typical state-of-practice processor: Intel Xeon E5645, we comprehensively characterize nineteen big data workloads included in *BigDataBench* with varying data inputs and have the following observation. First, in comparison with the traditional benchmarks: including *HPCC*, *PARSEC*, and *SPECCPU*, the floating point operation intensity of *BigDataBench* is two orders of magnitude lower than in the traditional benchmarks. Though for the big data applications, the average ratio of integer instructions to floating point instructions is about two orders of magnitude higher than in the traditional benchmarks, the average integer operation intensity of the big data applications is still in the same order of magnitude like those of the other benchmarks. Second, we observe that the volume of data input has non-negligible impact on micro-architecture events. For the worst cases, the number of MIPS (Million Instructions Per Second) of *Grep* has a 2.9 times gap between the baseline and the 32X data volume; the number of L3 cache MPKI of *K-means* has a 2.5 times gap between the baseline and the 32X data volume. This case may impose challenges for big data architecture research, since simulation-based approaches are widely used in architecture research and they are very time-consuming. Last but not least, corroborating the observations in *CloudSuite* [17] and *DCBench* [21] (which use smaller data inputs), we find that the numbers of L1I cache MPKI of the big data applications are higher than in the traditional benchmarks. We also find that L3 caches are effective for the big data applications, corroborating the observation in *DCBench* [21].

The rest of this paper is organized as follows. In Section 2, we discuss big data benchmarking requirements. Section 3 presents the related work. Section 4 summarizes our benchmarking methodology and decisions—-*BigDataBench*. Section 5 presents how to synthesize big

[1]OLTP is short for online transaction processing, referring to a class of information systems that facilitate and manage transaction-oriented applications with ACID (Atomicity, Consistency, Isolation, and Durability) support. Different from OLTP workloads, Cloud OLTP workloads do not need ACID support.

Table 1. Comparison of Big Data Benchmarking Efforts

Benchmark Efforts	Real-world data sets (Data Set Number)	Data scalability (Volume, Veracity)	Workloads variety	Software stacks	Objects to Test	Status
HiBench [20]	Unstructured text data (1)	Partial	Offline Analytics Realtime Analytics	Hadoop and Hive	Hadoop and Hive	Open Source
BigBench [19]	None	N/A	Offline Analytics	DBMS and Hadoop	DBMS and Hadoop	Proposal
AMP Benchmarks [5]	None	N/A	Realtime Analytics	Realtime analytic systems	Realtime analytic systems	Open Source
YCSB [15]	None	N/A	Online Services	NoSQL systems	NoSQL systems	Open Source
LinkBench [12]	Unstructured graph data (1)	Partial	Online Services	Graph database	Graph database	Open Source
CouldSuite [17]	Unstructured text data (1)	Partial	Online Services Offline Analytics	NoSQL systems, Hadoop, GraphLab	Architectures	Open Source

BigDataBench	Unstructured text data (1)
	Semi-structured text data (1)
	Unstructured graph data (2)
	Structured table data (1)
	Semi-structured table data (1)
		Total	Online Services
			Offline Analytics
			Realtime Analytics
				NoSQL systems, DBMS,
				Realtime Analytics
				Offline Analytics systems
					Systems and architecture;
					NoSQL systems;
					Different analytics systems
						Open Source

data while preserving characteristics of real-world data sets. In Section 6, we characterize *BigDataBench*. Finally, we draw the conclusion in Section 7.

2. Big Data Benchmarking Requirements

This section discusses big data benchmarking requirements.

(1) Measuring and comparing big data systems and architecture. First of all, the purpose of big data benchmarks is to measure, evaluate, and compare big data systems and architecture in terms of user concerns, e.g., performance, energy efficiency, and cost effectiveness. Considering the broad use cases of big data systems, for the sake of fairness, a big data benchmark suite candidate must cover not only broad application scenarios, but also diverse and representative real-world data sets.

(2) Being data-centric. Big data are characterized in four dimensions called ”4V” [14, 9]. *Volume* means big data systems need to be able to handle a large volume of data, e.g., PB. *Variety* refers to the capability of processing data of different types, e.g., un-structured, semi-structured, structured data, and different sources, e.g., text and graph data. *Velocity* refers to the ability of dealing with regularly or irregularly refreshed data. Additionally, a fourth V ”*veracity*” is added by IBM data scientists [9]. Veracity concerns the uncertainty of data, indicating that raw data characteristics must be preserved in processing or synthesizing big data.

(3) Diverse and representative workloads. The rapid development of data volume and variety makes big data applications increasingly diverse, and innovative application domains are continuously emerging. Big data workloads chosen in the benchmark suite should reflect diversity of application scenarios, and include workloads of different types so that the systems and architecture researchers could obtain the comprehensive workload characteristics of big data, which provides useful guidance for the systems design and optimization.

(4) Covering representative software stacks. Innovative software stacks are developed for specific user concerns. For examples, for online services, being latency-sensitivity is of vital importance. The influence of software stacks to big data workloads should not be neglected, so covering representative software stacks is of great necessity for both systems and architecture research.

(5) State-of-the-art techniques. In big data applications, workloads change frequently. Meanwhile, rapid evolution of big data systems brings great opportunities for emerging techniques, and a big data benchmark suite candidate should keep in pace with the improvements of the underlying systems. So a big data benchmark suite candidate should include emerging techniques in different domains. In addition, it should be extensible for future changes.

(6) Usability. The complexity of big data systems in terms of application scenarios, data sets, workloads, and software stacks prevents ordinary users from easily using big data benchmarks, so its usability is of great importance. It is required that the benchmarks should be easy to deploy, configure, and run, and the performance data should be easy to obtain.

3 Related work

We summarize the major benchmarking efforts for big data and compare them against BigDataBench in Table 1. The focus of most of the state-of-the-art big data benchmark efforts is evaluating specific types of applications or system software stacks, and hence not qualified for measuring big data systems and architectures, which are widely used in broad application scenarios.

Pavlo et al. [24] presented a micro benchmark for big data analytics. It compared Hadoop-based analytics to a row-based RDBMS system and a column-based RDBMS one. It is the Spark [30] and Shark [16] systems that inspire the AMP Lab big data benchmarks [5], which targets real-time analytic. This effort follows the benchmarking methodology in [24]. The benchmarks not only have a limited coverage of workloads, but also cover only table data. Its object under test is restricted to realtime analytics frameworks. HiBench [20] is a benchmark suite for Hadoop MapReduce and Hive. It covers incomplete data types and software stacks. GridMix [2] is a benchmark specially designed for Hadoop MapReduce, which includes only micro benchmarks for text data.

Internet services players also try to develop their benchmark suites. Yahoo! released their cloud benchmark specially for data storage systems, i.e, YCSB [15]. Having its root in cloud computing, YCSB is mainly for simple online service workloads—-so called "Cloud OLTP" workloads. Armstrong et al. [12] characterized the social graph data and database workloads for Facebook's social network, and presented the motivation, design, and implementation of LinkBench, a database benchmark that reflects real-world database workloads for social network applications. The TeraSort or GraySort benchmark [10] considers the performance and cost involved in sorting a large number of 100-byte records, and its workload is not sufficient to cover the various needs of big data processing. TPC-DS is TPC's latest decision support benchmark, covering complex relational queries for decision support. TPC-DS handles some aspects of big data like volume and velocity. Still, it lacks key data types like semi-structured and unstructured data and key applications types like realtime analytics. BigBench [19] is the recent effort towards designing big data benchmarks. BigBench focuses on big data offline analytics, thus adopting TPC-DS as the basis and adding atop new data types like semi-/un-structured data, as well as non-relational workloads. Although BigBench has a complete coverage of data types, its object under test is DBMS and MapReduce systems that claim to provide big data solutions, leading to partial coverage of software stacks. Furthermore, currently, it is not open-source for easy usage and adoption.

Recently, architecture communities also proposed *CloudSuite* [17] for scale-out cloud workloads, and *DCBench* [21] for datacenter workloads. Those efforts include small data sets, e.g., only 4.5 GB for *Naive Bayes* reported in *CloudSuite* [17]. Moreover, they fail to include diversity of real-world data sets and workloads. For example, for both *CloudSuite* and *DCBench*, realtime big data analytics workloads are not included, while they are very important emerging big data workloads. Moreover, they paid little attention to how to generate diversity of scalable big data sets (volume) while keeping their veracity.

4 Our Benchmarking Methodology and Decisions

This section presents our methodology and decisions on *BigDataBench*.

4.1 Our Benchmarking Methodology

In this paper, we consider all the big data benchmarking requirements mentioned in Section 2 based on a solid-founded methodology as shown in Figure. 1.

As there are many emerging big data applications, we take an incremental and iterative approach in stead of a top-down approach. First of all, we investigate the dominant application domains of internet services—an important class of big data applications according to widely acceptable metrics—the number of page views and daily visitors. According to the analysis in [3], 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. And then, we pay attention to typical data sets and big data workloads in the three application domains.

We consider data diversity in terms of both data types and data sources, and pay equal attention to structured, semi-structured, and unstructured data. Further, we single out three important data sources in the dominant application domains of internet services, including *text data*, on which the maximum amount of analytics and queries are performed in search engines [29], *graph data* (the maximum amount in social networks), and table data (the maximum amount in e-commerce). Other important data sources, e.g., multimedia data, will be continuously added. Furthermore, we propose novel data generation tools meeting with the requirements of data volume, variety, velocity, and veracity.

To cover diverse and representative workloads, we classify big data applications into three types from the users perspective: online services, offline analytics, and realtime analytics. An online service is very latency-sensitive, and for each request, comparatively simple operations are performed for delivering responses to end users immediately. For offline analytics, complex computations are performed on big data with long latency. While for realtime analytics, end users want to obtain analytic results in an interactive manner. We pay equal attention to three application types. Furthermore, we choose typical workloads from two dimensions: representative operations and algorithms from typical application scenarios, widely-used and state-of-the-art software stacks for three application types, respectively.

4.2 Chosen Data Sets

As analyzed in the big data benchmarking requirements, the data sets should be diverse and representative in terms of both data types and sources. After investigating three application domains, we collect six representative *real-world*

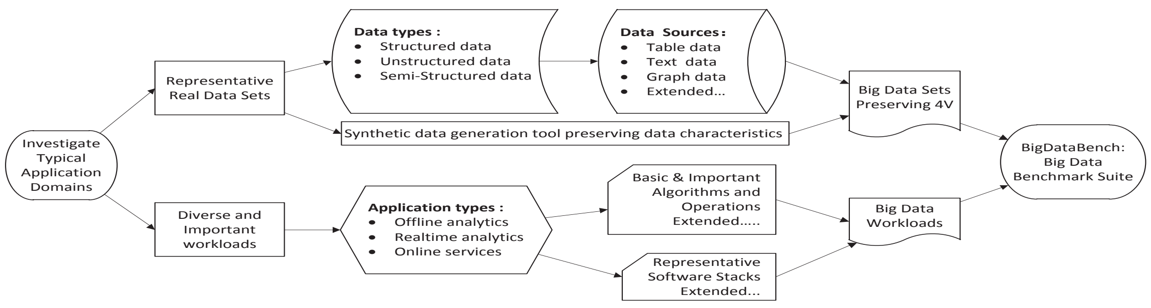

Investigate Typical Application Domains
Representative Real Data Sets
Diverse and Important workloads
Data types :
Structured data
Unstructured data
Semi-Structured data
Data Sources:
Table data
Text data
Graph data
Extended...
Synthetic data generation tool preserving data characteristics
Big Data Sets Preserving 4V
Application types :
Offline analytics
Realtime analytics
Online services
Basic & Important Algorithms and Operations Extended.....
Representative Software Stacks Extended...
Big Data Workloads
BigDataBench: Big Data Benchmark Suite

Figure 1. BigDataBench Methodology.

data sets. Our chosen data sets are diverse in three dimensions: data types, data sources, and application domains. Table 2 shows the characteristics of six real-world data sets. The original data set sizes are not necessarily scaled to the hardware and software to be tested. We need to scale the volume of the data sets while keeping their veracity, which we discuss in Section 5.

Table 2. The summary of real-world 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	Table 1: 4 columns, 38658 rows. Table 2: 6 columns, 242735 rows
6	ProfSearch Person Resumés	278956 resumés

Wikipedia Entries [11]. The Wikipedia data set is unstructured, consisting of 4,300,000 English articles. Four workloads use this data set, including *Sort*, *Grep*, *WordCount* and *Index*.

Amazon Movie Reviews [4]. This data set is semi-structured, consisting of 7,911,684 reviews on 889,176 movies by 253,059 users. The data span from Aug 1997 to Oct 2012. Two workloads use this data set, including *Naive Bayes* for sentiment classification, and *Collaborative Filtering (in short, CF)*– a typical recommendation algorithm.

Google Web Graph (Directed graph)[8]. This data set is unstructured, containing 875713 nodes representing web pages and 5105039 edges representing the links between web pages. This data set is released by Google as a part of Google Programming Contest. We use it for *PageRank*.

Facebook Social Graph (Undirected graph) [7]. This data set contains 4039 nodes, which represent users, and 88234 edges, which represent friendship between users. The data set is used for the graph mining workload– *Connected Components, in short (CC)*.

Table 3. Schema of E-commerce Transaction Data

ORDER	ITEM
ORDER_ID INT	ITEM_ID INT
BUYER_ID INT	ORDER_ID INT
CREATE_DATE DATE	GOODS_ID INT
	GOODS_NUMBER NUMBER(10,2)
	GOODS_PRICE NUMBER(10,2)
	GOODS_AMOUNT NUMBER(14,6)

E-commerce Transaction Data. This data set is from an e-commerce web site, which we keep anonymous by request. The data set is structured, consisting of two tables: ORDER and order ITEM. The details are shown in Table 3. This data set is used for the relational queries workloads.

***ProfSearch* Person Resumés**. This data set is from a vertical search engine for scientists developed by ourselves, and its web site is `http://prof.ict.ac.cn`. The data set is semi-structured, consisting of 278956 resumés automatically extracted from 20,000,000 web pages of about 200 universities and research institutions. This data set is used for "Cloud OLTP" workloads.

We plan to add other real-world data sets to investigate the impact of different data sets on the same workloads.

4.3 Chosen Workloads

We choose the *BigDataBench* workloads with the following considerations: 1) Paying equal attention to different types of applications: online service, real-time analytics, and offline analytics; 2) Covering workloads in diverse and representative application scenarios ; 3) Including different data sources: text, graph, and table data; 4) Covering the representative big data software stacks.

In total, we choose 19 big data benchmarks. Table 4 presents *BigDataBench* from perspectives of application scenarios, operations/ algorithms, data types, data sources, software stacks, and application types. For some end users, they may just pay attention to big data application of a spe-

Table 4. The Summary of BigDataBench.

Application Scenarios	Application Type	Workloads	Data types	Data source	Software Stacks
Micro Benchmarks	Offline Analytics	Sort	Unstructured	Text	Hadoop, Spark, MPI
		Grep			
		WordCount			
		BFS		Graph	
Basic Datastore Operations ("Cloud OLTP"	Online Service	Read	Semi-structured	Table	Hbase, Cassandra, MongoDB, MySQL
		Write			
		Scan			
Relational Query	Realtime Analytics	Select Query	Structured	Table	Impala, MySQL, Hive, Shark
		Aggregate Query			
		Join Query			
Search Engine	Online Services	Nutch Server	Un-structured	Text	Hadoop
	Offline Analytics	Index			
		PageRank		Graph	Hadoop, Spark, MPI
Social Network	Online Services	Olio Server	Un-structured	Graph	Apache+MySQL
	Offline Analytics	Kmeans			Hadoop, Spark, MPI
		Connected Components (CC)			
E-commerce	Online Services	Rubis Server	Structured	Table	Apache+JBoss+MySQL
	Offline Analytics	Collaborative Filtering (CF)			Hadoop, Spark, MPI
		Naive Bayes	Semi-structured	Text	

cific type. For example, they want to perform an apples-to-apples comparison of software stacks for realtime analytics. They only need to choose benchmarks with the type of real-time analytics. But if the users want to measure or compare big data systems and architecture, we suggest they cover all benchmarks.

To cover diverse and representative workloads, we include important workloads from three important application domains: search engines, social networks, and e-commence. In addition, we include micro benchmarks for different data sources, "Cloud OLTP" workloads, and relational queries workloads, since they are fundamental and pervasive. The workload details are shown in the user manual available from [6].

For different types of big data applications, we also include widely-used and state-of-the-art system software stacks. For example, for offline analytics, we include MapReduce, and MPI, which is widely used in HPC communities. We also include Spark, which is best for iterative computation. Spark supports in-memory computing, letting it query data faster than disk-based engines like MapReduce-based systems. *Most of the benchmarks in the current release [6], are implemented with Hadoop. But we plan to release other implementations, e.g., MPI, Spark.*

5 Synthetic Data Generation Approaches and Tools

How to obtain big data is an essential issue for big data benchmarking. A natural idea to solve these problems is to generate synthetic data while keeping the significant features of real data. Margo Seltzer et al. [25] pointed that if we want to produce performance numbers that are meaningful in the context of real applications, we need use application-specific benchmarks. Application-specific benchmarking would need application-specific data generation tools, which synthetically scale up real-world data sets while keeping their data characteristics [26]. That is to say, for different data types and sources, we need to propose different approaches to synthesizing big data.

Since the specific applications and data are diverse, the task of synthesizing big data on the basis of real-world data is nontrivial. The data generation procedure in our benchmark suite is as follows: First, we should have several representative real-world data sets which are application-specific. And then, we estimate the parameters of the data models using the real-world data. Finally we generate synthetic data according to the data models and parameters, which are obtained from real-world data.

We develop *Big Data Generator Suite* (in short, *BDGS*)– a comprehensive tool–to generate synthetic big data preserving the 4V properties. The data generators are designed for a wide class of application domains (search engine, e-commence, and social network), and will be extended for other application domains. We demonstrate its effectiveness by developing data generators based on six real life data sets that cover three representative data types (structured, semi-structured, and unstructured data), three data sources (text, graph, and table). Each data generator can produce synthetic data sets, and its data format conversion tools can transform these data sets into an appropriate format capable of being used as the inputs of a specific workload. Users can specify their preferred data size. In theory, the data size limit can only be bounded by the storage size and the *BDGS* parallelism in terms of the nodes and its running time. The details of generating text, graph, and table data can be found at [23].

Table 5. Node configuration details of Xeon E5645

CPU Type		Intel CPU Core	
Intel ®Xeon E5645		6 cores@2.40G	
L1 DCache	L1 ICache	L2 Cache	L3 Cache
6 × 32 KB	6 × 32 KB	6 × 256 KB	12MB

6 Workload Characterization Experiments

In this section, we present our experiment configurations and methodology, the impact of the data volume on micro-architecture events, and workload characterization of big data benchmarks, respectively.

6.1 Experiments Configurations and Methodology

We run a series of workload characterization experiments using *BigDataBench* to obtain insights for architectural studies. Currently, we choose Hadoop as the basic software stack. Above Hadoop, HBase and Nutch are also tested. Besides, MPICH2 and Rubis are deployed for understanding different workloads. In the near future, we will study the impact of different implementations on workload characterization using other analytic frameworks.

For the same big data application, the scale of the system running big data applications is mainly decided by the size of data input. For the current experiments, the maximum data input is about 1 TB, and we deploy the big data workloads on the system with a matching scale—14 nodes. Please note that with our data generation tools in *BigDataBench*, users can specify a larger data input size to scale up the real-world data, and hence need a larger system. On our testbed, each node has two Xeon E5645 processors equipped with 16 GB memory and 8 TB disk. The detailed configuration of each node is listed in Table 5. Please note that in the rest experiments, hyperthreading is enabled on our testbed. The operating system is Centos 5.5 with Linux kernel 2.6.34. The Hadoop distribution is 1.0.2 with Java version 1.6. The HBase, Hive, MPICH2, Nutch, and Rubis distribution is 0.94.5, 0.9, 1.5, 1.1, 5.0, respectively. With regard to the input data, we vary the size from 32GB to 1TB for the analytics workloads. As it has large data complexity, the input data for the graph-related workloads like *BFS*, *CC*, and *CF* workloads are measured in terms of the set of vertices, while those of *Index* and *PageRank* workloads are in terms of Web pages. We also vary the request number from 100 requests per second to 3200 requests per second for all service workloads. Table 6 shows the workload summary.

6.1.1 Experiment Methodology

Modern processors provide hardware performance counters to support micro-architecture level profiling. We use Perf, a Linux profiling tool, to collect about 20 events whose numbers and unit masks can be found in the Intel Developer's Manual. In addition, we access the proc file system to collect OS-level performance data. We collect performance data after a ramp up period, which is about 30 seconds.

Table 6. Workloads in experiments

ID	Workloads	Software Stack	Input size
1	Sort	Hadoop	32 ×(1,..,32) GB data
2	Grep	Hadoop	32 ×(1,..,32)GB data
3	WordCount	Hadoop	32 ×(1,..,32)GB data
4	BFS	MPI	2^{15}×(1,..,32) vertex
5	Read	Hbase	32 ×(1,..,32) GB data
6	Write	Hbase	32 ×(1,..,32)GB data
7	Scan	Hbase	32 ×(1,..,32) GB data
8	Select Query	Hive	32 ×(1,..,32) GB data
9	Aggregate Query	Hive	32 ×(1,..,32)GB data
10	Join Query	Hive	32 ×(1,..,32)GB data
11	Nutch server	Hadoop	100 ×(1,..,32) req/s
12	PageRank	Hadoop	10^6×(1,..,32) pages
13	Index	Hadoop	10^6×(1,..,32) pages
14	Olio Server	MySQL	100 ×(1,..,32) req/s
15	K-means	Hadoop	32GB ×(1,..,32) data
16	CC	Hadoop	2^{15}×(1,..,32) vertex
17	Rubis Server	MySQL	100 ×(1,..,32) req/s
18	CF	Hadoop	2^{15}×(1,..,32) vertex
19	Naive Bayes	Hadoop	32 ×(1,..,32) GB data

6.1.2 Metrics

Two categories of metrics are used for evaluation. The first category of metrics are *user-perceivable metrics*, which can be conveniently observed and understood by users. The second ones are *architectural metrics*, which are mainly exploited by architecture research. In the first category of metrics, we choose three measuring units for different workloads, respectively. The number of processed requests per second (*RPS* in short) is used to measure the throughput of online service workloads. In addition, we also care about latency. The number of operations per second (*OPS* in short) is used to evaluate "Cloud OLTP" workloads. And, the data processed per second (*DPS* in short) is used for analytic workloads[22]. *DPS* is defined as the input data size divided by the total processing time. In comparison with the metrics like the processed jobs or tasks per time unit, *DPS* is much more relevant to the data processing capability of the system which users concern[22]. The second category is chosen to compare performances under different workloads. Though the user-perceivable metrics can help evaluating different workloads in the same category, it is impossible to compare performances of workloads from different categories, e.g., a database server and a MapReduce workload. Hence, the uniform architecture metrics are necessary. Since no heterogeneous platform is involved in the experiments, we choose the widely accepted performance metrics in the architecture research, e.g., MIPS, and cache MPKI.

6.1.3 The Other Benchmarks Setup

For *SPEC CPU2006*, we run the official applications with the first reference input, and report the average results into two groups: integer benchmarks (*SPECINT*) and floating point benchmarks (*SPECFP*). *HPCC* is a representative HPC benchmark suite, and we run *HPCC* with version 1.4. We run all seven benchmarks, including *HPL*, *STREAM*, *PTRANS*, *RandomAccess*, *DGEMM*, *FFT*, and *COMM*. *PARSEC* is a benchmark suite composed of multi-threaded programs, and we deploy *PARSEC* 3.0 Beta Release. We run all 12 benchmarks with native input data sets and use gcc with version 4.1.2 to compile them.

6.2 The Implication of Data Volume for Architecture Research

Intuitively, data input should be highly relevant to big data workloads characterization. Specifically, the size of data input should be relevant to micro-architectural characteristics. In this subsection, we pay attention to an important issue—*what amount of data qualifies for being called big data from a perspective of workload characterization*? This issue is very important and interesting, because simulation is the basic approach for architecture research, but it is very time-consuming. Bigger input data size would significantly increase the run time of a program, especially on the simulation platform. If there is no obvious difference between large and small data inputs in terms of micro-architectural events, the simulation-based approaches using small data sets can still be valid for architecture research. As different workloads have different input data types and sizes, we set the minimum data scales in Table 6 as the baseline data inputs, e.g., 32 GB for *Sort*, 1000000 pages for *PageRank*, and 100 requests per second for *Nutch Server*. On the baseline data input, we scale up the data size by 4, 8, 16 and 32 times, respectively.

There are hundreds of micro-architectural events in modern processors. For better readability and data presentation, we only report the numbers of MIPS and L3 cache MPKI. Figure 3-1 demonstrates MIPS numbers of each workload with different data scales. From Figure 3-1, we find that for different workloads, the instruction executing behaviors exhibit different trends as the data volume increases. For example, MIPS numbers of *Grep* and *WordCount* increase after the 16 times baseline, while for some other workloads, they tend to be stable after the data volume increases to certain thresholds. The cache behavior metrics also exhibit a similar phenomenon as the MIPS metric does. As important as L3 cache misses are–a single one can cause hundreds of cycles of latency–we track this value for different workloads under different configurations. In Figure 2, for a workload, we call the data input on which the system achieves the best performance as the *large input* for a workload, and the baseline as the *small input*. From Figure 2, we can see

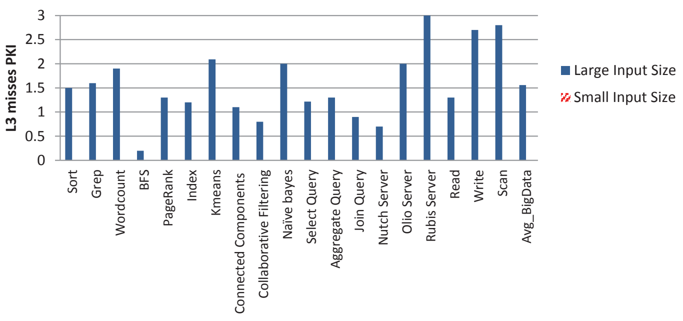
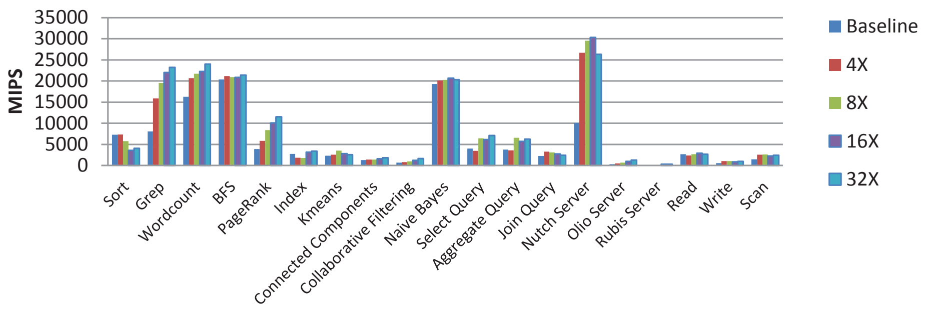


Figure 2. L3 cache MPKI of different configurations in big data workloads.

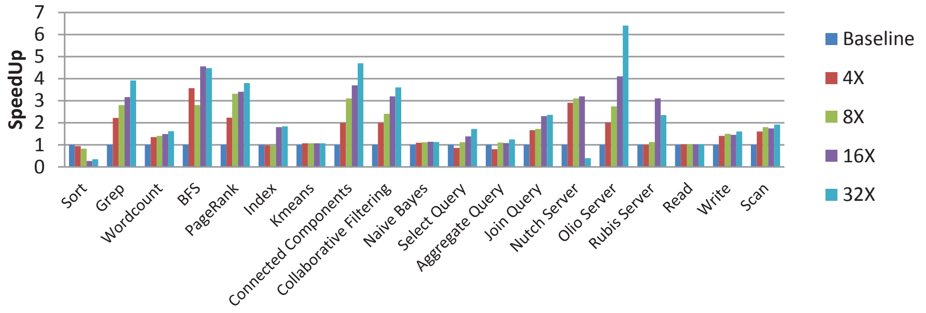
that some workloads have lower number of L3 cache MPKI on the large configuration, e.g., *Sort*, while some have higher number of L3 cache MPKI on the large configuration, e.g., *Grep*. There are the other workloads showing no obvious difference under the two configurations, e.g., *Index*. *K-means* has the largest difference under the two configurations, and the number of L3 cache MPKI is 0.8 and 2 for the small and large data inputs, respectively, which shows different data inputs can result in significantly different cache performance evaluation results.

To help understand the micro-architecture events, we also stress test the cluster system with increasing data scale, and report the user-perceivable performance number as mentioned in Section 6.1.2. Because workloads in Table 6 have different metrics, we set the performance number from the experiments with the baseline data inputs as the baseline, and then normalize the collected results for each workload with varying data inputs over the baseline number. For example, the performance number of *Workload A* for the baseline input is x and that for 4 $\times$ baseline input is y, and then for *Workload A* we normalize the performance number for the baseline input and 4 $\times$ baseline input as one and ($y \div x$), respectively. Figure 3-2 reports the normalized performance numbers of each *BigDataBench* workload with different data volumes. Please note that the performance of *Sort* degrades with increased data size in Figure 3-2 because *Sort* is an I/O intensive workload when the memory cannot hold all its input data. Besides, the larger data sizes demand more I/O operations like shuffling and disk accesses. Worse still, the network communication involved in data shuffling causes congestion, thus impairing performance.

[**Lessons Learned**]. As the above figures show, we find that different big data workloads have different performance trends as the data scale increases. This is the reason we believe that the workloads that only cover a specific application scenario are not sufficient to evaluate big data systems and architecture. Second, architectural metrics are closely related to input data volumes and vary for different work-



3-1 MIPS of different workloads with different data scale



3-2 Speedup of different workloads with different data scale

Figure 3. Performance data vary with different data input sizes.

loads, and data volume has non-negligible impact on workload characterization. For example, the MIPS number of *Grep* has a 2.9 times gap between the baseline and 32X data volume; the L3 cache MPKI of *K-means* has a 2.5 times gap between the baseline and 32X data volume. This result implies that using only simple applications with small data sets is not sufficient for big data systems and architecture research, which may impose great challenges.

6.3 Workload Characterization

This section mainly focuses on characterizing operation intensity and cache behaviors of big data workloads.

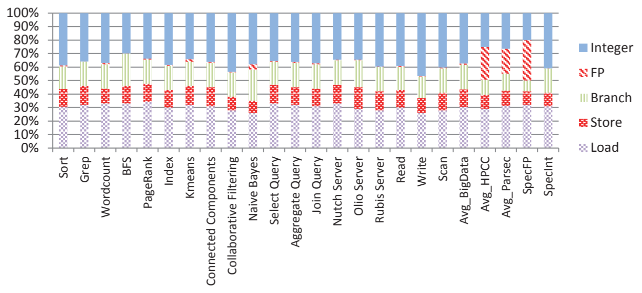


Figure 4. Instruction Breakdown.

6.3.1 Measuring operation intensity.

In order to characterize instruction behaviors, first, we breakdown the execution instructions. As shown in Figure 4, big data workloads have the distinct feature that the ratio of integer instructions to floating-point instructions is very high. On Intel Xeon E5645, the average ratio is 75. The maximum is 179 (*Grep*), and the minimum is 10 (*Bayes*). For comparison, these ratios for *PARSEC*, *HPCC* and *SPECFP* are very low, on the average 1.4, 1.0, and 0.67, respectively. The ratio for SPECINT (on the average 409) is the highest, because it is intended to evaluate integer operations of processors. From this perspective, we can conclude that the big data workloads significantly differ from the traditional benchmarks like *HPCC*, *PARSEC*, and *SPECCFP*. *Please note that the reported numbers may deviate across different processors*. For example, Intel processors uses different generations of SSE (Streaming SIMD Extensions), which introduces both scalar and packed floating point instructions.

Furthermore, for each workload, we calculate *the ratio of computation to memory access* to measure the operation intensity. *Floating point or integer operation intensity* is defined as the total number of (floating point or integer) instructions divided by the total number of memory accesses in terms of bytes in a run of the workload [28]. For example, in a run of *program A*, it has *n* floating point instructions and *m* bytes of memory accesses, so the operation intensity of *program A* is $(n \div m)$. Since the memory hierarchy would impact the memory access performance significantly, for comparison, we report experiments on two state-of-practice processors: the Xeon E5310 and the Xeon E5460, respectively. The Xeon E5310 is equipped with only two levels of caches, while the Xeon E5645 is equipped with three levels of caches. The configuration of the Xeon E5310 is shown in Table. 7.

Table 7. Configuration details of Xeon E5310.

CPU Type		Intel CPU Core	
Intel ®Xeon E5310		4 cores@1.60G	
L1 DCache	L1 ICache	L2 Cache	L3 Cache
4 × 32 KB	4 × 32 KB	2 × 4MB	None

In Figure 5-1, we can see that big data workloads have very low floating point operation intensities, and the average number of *BigDataBench* is 0.007 on the Xeon E5310, and 0.05 on the Xeon E5645, respectively. However, the number of *PARSEC*, *HPCC*, and *SPCECFP* is higher as 1.1, 0.37, 0.34 on the Xeon E5310, and 1.2, 3.3, 1.4 on the Xeon E5645, respectively. *SPECINT* is an exception with the number closing to 0. On the average, *HPCC* and *PARSEC* have high operation intensity because of their computing-intensive kernels, and *SPECFP* has relatively high operation intensity for it is oriented for floating point operations. In summary, the floating point operation intensity of *BigDataBench* is two orders of magnitude lower than in the traditional workloads on the Xeon E5310, and Xeon E5645, respectively. The reason the floating point operation intensity of *BigDataBench* on E5645 is higher than on E5310 can be partly explained by the fact that L3 caches are effective in decreasing the memory access traffic, which will be further analyzed in next subsection.

Though the average ratio of integer instructions to floating-point ones of big data workloads is about two orders of magnitude larger than in the other benchmarks, the average integer operation intensity of big data workloads is in the same order of magnitude like those of the other benchmarks. As shown in Figure 5-2, the average integer operation intensity of *BigDataBench*, *PARSEC*, *HPCC*, *SPECFP* and *SPECINT* is 0.5, 1.5, 0.38, 0.23, 0.46 on the Xeon E5310 and 1.8, 1.4, 1.1, 0.2, 2.4 on the Xeon E5645, respectively.

[**Lessons Learned**]. In conclusion, we can say that in comparison with the traditional benchmarks, the big data workloads in *BigDataBench* have low ratios of computation to memory accesses. The above phenomenon can be explained from two aspects. First, big data processing heavily relies upon memory accesses. Second, big data workloads must process large volume of data, and hence most big data workloads adopt simple algorithms with low computing complexity. In *BigDataBench*, they range from

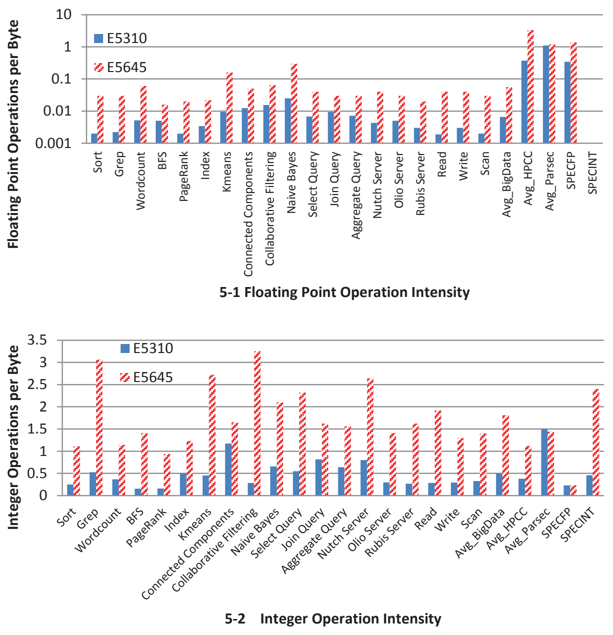


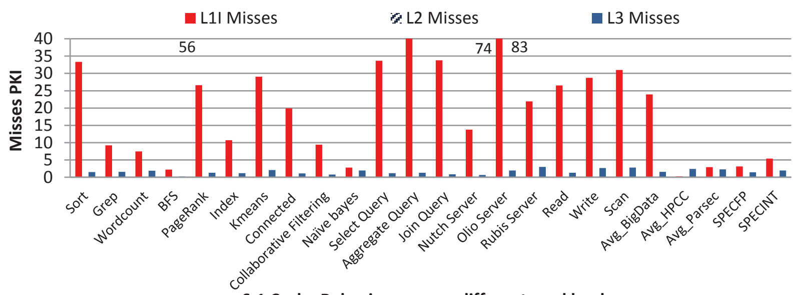
Figure 5. Operation Intensity on Intel Xeon E5310 and E5645.

O(n) to *O(n*lgn)*. In comparison, most *HPCC* or *PARSEC* workloads have higher computing complexity, ranging from *O(n*lgn)* to $O(n^3)$. We can make the conclusion that big data workloads have higher demand for data movements than instruction executions. The state-of-practice processor is not efficient for big data workloads. Rather, we believe that for these workloads the floating-point unit is over-provisioned.

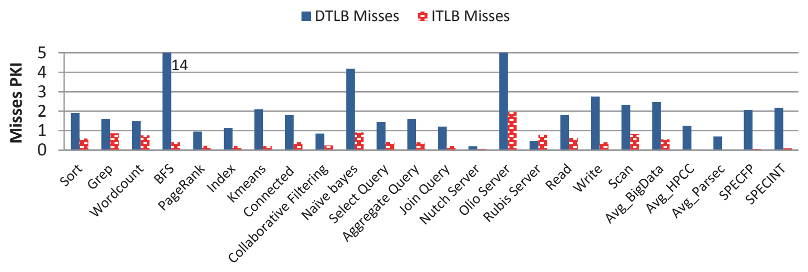
6.3.2 Memory Hierarchy Analysis.

Finally, we show the operation intensity of the big data workloads is low, and we want to further investigate their cache behaviors. In this subsection, we report L3 cache MPKI, L2 cache MPKI, L1 instruction MPKI, instruction TLB MPKI, and data TLB MPKI, respectively. We leave out L1D cache MPKI because its miss penalty can be hidden by the out-of-order pipeline. From Figure 6, we observe that the cache behaviors of BigDataBench have three significant differences from the traditional benchmarks as follows:

First, the average L1I cache MPKI of *BigDataBench* is at least four times higher than in the traditional benchmarks. The average L1I cache MPKI of *BigDataBench* is 23, while that of *HPCC*, *PARSEC*, *SPECFP*, and *SPECINT* is 0.3, 2.9, 3.1, and 5.4, respectively. This observation corroborates the ones in CloudSuite and DCBench. The possible main factors leading to the high L1I cache MPKI are the huge code size and deep software stack of the big data workloads. Second, the average L2 cache MPKI of *Big-*



6-1 Cache Behaviors among different workloads



6-2 TLB Behaviors among different workloads

Figure 6. Memory hierarchy behaviors among different workloads.

DataBench is higher than in the traditional workloads. The average L2 cache MPKI of *BigDataBench* is 21, while that for *HPCC*, *PARSEC*, *SPECFP*, and *SPECINT* is 4.8, 5.1, 14, and 16, respectively. Among *BigDataBench*, most of the online service workloads have the higher L2 cache MPKI (on the average, 40) except *Nutch server* (4.1), while most of the (offline and realtime) analytics workloads have the lower L2 cache MPKI (on the average, 13) except *BFS* (56). Third, the average L3 cache MPKI of *BigDataBench* is 1.5, while the average number of *HPCC*, *PARSEC*, *SPECFP*, and *SPECINT* is 2.4, 2.3, 1.4 and 1.9, respectively. This observation shows that the LLC (L3) caches of the processors (Xeon E5645) on our testbed are efficient for the big data workloads, corroborating the observation in *DCBench*. The efficiency of L3 caches also can explain why the floating point intensity of *BigDataBench* on the Xeon E5645 is higher than on the Xeon E5310 (only two levels of caches).

The TLB behaviors are shown in Figure. 6-2. First, the average number of ITLB MPKI of *BigDataBench* is higher than in the traditional workloads. The average number of ITLB MPKI of *BigDataBench* is 0.54, while that of *H-PCC*, *PARSEC*, *SPECFP*, and *SPECINT* is 0.006, 0.005, 0.06, and 0.08, respectively. The more ITLB MPKI of *BigDataBench* may be caused by the complex third party libraries and deep software stacks of the big data workloads. Second, the average number of DTLB MPKI of *BigDataBench* is also higher than in the traditional workloads. The average number of DTLB MPKI of *BigDataBench* is 2.5, while that of *HPCC*, *PARSEC*, *SPECFP*, and *SPECINT* is 1.2, 0.7, 2, and 2.1, respectively. And we can also find that the numbers of DTLB MPKI of the big data workloads range from 0.2 (*Nutch server*) to 14 (*BFS*). The diversity of DTLB behaviors reflects the data access patterns are diverse in big data workloads, which proves that diverse workloads should be included in big data benchmarks.

[**Lessons Learned**]. On a typical state-of-practice processor: Intel Xeon E5645, we find that L3 caches of the processor are efficient for the big data workloads, which indicates multi-core CPU design should pay more attention to area and energy efficiency of caches for the big data applications. The high number of L1I cache MPKI implies that better L1I cache performance is demanded for the big data workloads. We conjecture that the deep software stacks of the big data workloads are the root causes of high frond-end stalls. We are planning further investigation into this phenomenon by changing the software stacks under test, e.g., replacing MapReduce with MPI.

7 Conclusion

In this paper, we presented our joint research efforts with several industrial partners on big data benchmarking. Our methodology is from real systems, covering not only broad application scenarios but also diverse and representative real-world data sets. We proposed an innovative data generation methodology and tool to generate scalable volumes of big data keeping the 4*V* properties. Last, we chose and developed nineteen big data benchmarks from dimensions of application scenarios, operations/ algorithms, data types, data sources, software stacks, and application types. Also, we reported the workloads characterization results of big data as follows: first, in comparison with the traditional benchmarks, the big data workloads have very low operation intensity. Second, the volume of data input has non-negligible impact on micro-architecture characteristics of big data workloads, so architecture research using only simple applications and small data sets is not sufficient for big data scenarios. Last but not least, on a typical state-of-practice processor: Intel Xeon E5645, we find that for the big data workloads the LLC of the processor is effective and better L1I cache performance is demanded as the big data workloads suffer high L1I cache MPKI.

8 Acknowledgements

We are very grateful to the anonymous reviewers and our shepherd, Dr. Michael D. Powell. This work is supported by Chinese 973 projects (Grants No. 2011CB302502 and 2014CB340402), NSFC projects (Grants No. 60933003 and 61202075) and the BNSF project (Grant No.4133081).

References

[1] http://www-01.ibm.com/software/data/bigdata/.

[2] http://hadoop.apache.org/mapreduce/docs/current/gridmix.html.

[3] Alexa website. http://www.alexa.com/topsites/global.

[4] Amazon movie reviews. http://snap.stanford.edu/data/web-Amazon.html.

[5] Amp lab big data benchmark. https://amplab.cs.berkeley.edu/benchmark/.

[6] Bigdatabench. http://prof.ict.ac.cn/BigDataBench/.

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

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

[9] ibmbigdatahub. http://www.ibmbigdatahub.com/infographic/four-vs-big-data.

[10] Sort benchmark home page. http://sortbenchmark.org/.

[11] wikipedia. http://en.wikipedia.org.

[12] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan. Linkbench: a database benchmark based on the facebook social graph. 2013.

[13] L. A. Barroso and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 4(1):1–108, 2009.

[14] M. Beyer. Gartner says solving big data challenge involves more than just managing volumes of data. http://www.gartner.com/it/page.jsp?id=1731916.

[15] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, 2010.

[16] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: fast data analysis using coarse-grained distributed memory. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 689–692. ACM, 2012.

[17] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: A study of emerging workloads on modern hardware. *Architectural Support for Programming Languages and Operating Systems*, 2012.

[18] W. Gao, Y. Zhu, Z. Jia, C. Luo, L. Wang, J. Zhan, Y. He, S. Gong, X. Li, S. Zhang, and B. Qiu. Bigdatabench: a big data benchmark suite from web search engines. *The Third Workshop on Architectures and Systems for Big Data (ASBD 2013), in conjunction with ISCA 2013.*

[19] A. Ghazal, M. Hu, T. Rabl, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. Bigbench: Towards an industry standard benchmark for big data analytics. In *SIGMOD 2013*, 2013.

[20] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 41–51. IEEE, 2010.

[21] 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.

[22] C. Luo, J. Zhan, Z. Jia, L. Wang, G. Lu, L. Zhang, C. Xu, and N. Sun. Cloudrank-d: benchmarking and ranking cloud computing systems for data processing applications. *Frontiers of Computer Science*, 6(4):347–362, 2012.

[23] Z. Ming, C. Luo, W. Gao, R. Han, Q. Yang, L. Wang, and J. Zhan. Bdgs: A scalable big data generator suite in big data benchmarking. In *Arxiv*, 2014.

[24] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 165–178, 2009.

[25] M. Seltzer, D. Krinsky, K. Smith, and X. Zhang. The case for application-specific benchmarking. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 102–107. IEEE, 1999.

[26] Y. Tay. Data generation for application-specific benchmarking. *VLDB, Challenges and Visions*, 2011.

[27] P. Wang, D. Meng, J. Han, J. Zhan, B. Tu, X. Shi, and L. Wan. Transformer: a new paradigm for building data-parallel programming models. *Micro, IEEE*, 30(4):55–64, 2010.

[28] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[29] H. Xi, J. Zhan, Z. Jia, X. Hong, L. Wang, L. Zhang, N. Sun, and G. Lu. Characterization of real workloads of web search engines. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, volume 11, pages 15–25. IEEE, 2011.

[30] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

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