Discussion of BigBench: A Proposed Industry Standard Performance Benchmark for Big Data

Chaitanya Baru[11], Milind Bhandarkar[10], Carlo Curino[7], Manuel Danisch[1], Michael Frank[1], Bhaskar Gowda[6], Hans-Arno Jacobsen[8], Huang Jie[6], Dileep Kumar[3], Raghunath Nambiar[2], Meikel Poess[9], Francois Raab[5], Tilmann Rabl[1,8(✉)], Nishkam Ravi[3], Kai Sachs[12], Saptak Sen[4], Lan Yi[6], and Choonhan Youn[11]

[1] Bankmark, Passau, Germany
{manuel.danisch,michael.frank}@bankmark.de,
tilmann.rabl@utoronto.ca
[2] Cisco Systems, San Jose, USA
rnambiar@cisco.com
[3] Cloudera, Palo Alto, USA
{dkumar,nravi}@cloudera.com
[4] Hortonworks, Santa Clara, USA
[5] Infosizing, Manitou Springs, USA
francois@sizing.com
[6] Intel Corporation, Santa Clara, USA
{bhaskar.d.gowda,jie.huang,lan.yi}@intel.com
[7] Microsoft Corporation, Redmond, USA
ccurino@microsoft.com
[8] Middleware Systems Research Group, Toronto, Canada
jacobsen@eecg.toronto.edu
[9] Oracle Corporation, Redwood City, USA
meikel.poess@oracle.com
[10] Pivotal, Vancouver, Canada
mbhandarkar@gopivotal.com
[11] San Diego Supercomputer Center, La Jolla, USA
{baru,cyoun}@sdsc.edu
[12] SPEC Research Group, Gainesville, USA
kai.sachs@sap.com

Abstract. Enterprises perceive a huge opportunity in mining information that can be found in big data. New storage systems and processing paradigms are allowing for ever larger data sets to be collected and analyzed. The high demand for data analytics and rapid development in technologies has led to a sizable ecosystem of big data processing systems. However, the lack of established, standardized benchmarks makes it difficult for users to choose the appropriate systems that suit their requirements. To address this problem, we have developed the BigBench benchmark specification. BigBench is the first *end-to-end* big data analytics benchmark suite. In this paper, we present the BigBench benchmark and analyze the workload from technical as well as business point of view. We characterize the queries in the workload along different dimensions, according to their functional characteristics, and also analyze their

R. Nambiar and M. Poess (Eds.): TPCTC 2014, LNCS 8904, pp. 44–63, 2015.
DOI: 10.1007/978-3-319-15350-6_4

runtime behavior. Finally, we evaluate the suitability and relevance of the workload from the point of view of enterprise applications, and discuss potential extensions to the proposed specification in order to cover typical big data processing use cases.

1 Introduction

Enterprises everywhere appear to be reaching a tipping point with data. Large amounts of data are being accumulated; data continue to arrive from ever increasing number of sources, and at increasing rates; and most applications require integration of data from multiple heterogeneous sources. The data need to be queried and analyzed to support enterprise applications. Organizations view these data as a "natural resource" from which they can potentially extract significant value for the enterprise. Indeed, this phenomenon, referred to as "big data", is the driving force behind major commercial investments in hardware and software. In the current landscape of enterprise big data systems, two major architectures dominate the analytics market: parallel database systems and Hadoop-style batch-oriented systems. While there have been several studies that have attempted to compare and contrast these two approaches, what is lacking is a benchmark specification that can be used to objectively compare systems with each other. Furthermore, big data hardware and software vendors are rapidly evolving their systems to meet the applications needs and demands of these big data applications. In some cases, there is a common approach emerging, such as increased support for SQL-like functions, or better support for online query processing, rather than just batch processing. As vendors begin to incorporate similar features and compete in the same markets, it become essential to have objective benchmarks that can be used to compare system performance, as well as price/performance and energy consumption.

Thus far, due to lack of existing, accepted standards, vendors have been forced to run *ad hoc* benchmarks, or simple benchmarks which may not reflect the eventual workload encountered by the systems. Furthermore, they have not had to provide full disclosures regarding system performance. An industry standard will be able to address such shortcomings, thus improving the overall situation.

We propose **BigBench** as a first, important step in moving towards a set of rigorous benchmarks for big data systems. Similar to the well-known TPC benchmarks, BigBench is an "application-level" benchmark. It captures operations performed at an application level via SQL queries and data mining operations, rather than low level operations such as, say, file I/O, or performance of specific function such as sorting or graph traversal.

In this paper, we provide a detailed discussion of the BigBench specification, including the database and the workload. In the process of developing BigBench, we have obtained feedback from leading industry experts about the relevance as well as completeness of the workload. After a technical discussion of the benchmark and a discussion of sample runs on two different "small" and "large" platforms, we provide a summary of the feedback as well as ideas for

future extensions to the benchmark. We recognize that *Big Data* is a complex as well as evolving space. BigBench represents only the first step towards providing a systematic way of benchmarking big data systems. We expect that big data benchmarking will need to be an *agile* activity for the near-term future, in order to both keep pace with changing technological trends and the evolving application requirements in this area.

The paper is organized as follows. Section 2 describes benchmarking efforts and activity relevant to big data and to BigBench. Section 3 provides an overview of the BigBench benchmark, followed immediately by a description of the experiments performed on the small and large test platforms in Sect. 4. Section 5 summarizes the characteristics of the BigBench schema as well as the queries in the workload. Section 6 discusses the community feedback that was provided. Based on this, some possible future extensions to BigBench are presented in Sect. 7. Including a broad range of features within a single benchmark would likely make the benchmark unwieldy, difficult to understand, difficult and expensive to implement and, most important, difficult to interpret the results. Our goal is to capture community feedback, and use the information to develop a roadmap of big data benchmarks, rather than incorporating all features into a single unwieldy benchmark. Section 8 elaborates on the additional steps needed to make BigBench an industry standard benchmark, based on experience with benchmarks like the TPC. Finally, the paper concludes with Sect. 9.

2 Related Work

A number of efforts are currently underway for developing benchmarks for different aspects of big data systems. For example, *TPC-H* [14] and *TPC-DS* [12] benchmarks, developed by the Transaction Processing Performance Council, have been used for benchmarking big data systems. The TPC-H benchmark has been implemented in Hadoop, Pig, and Hive [5,18]. A subset of TPC-DS has been used to compare query performance with implementations using Impala and Hive. However, while they have been used for measuring performance of big data systems, both TPC-H and TPC-DS are "pure SQL" benchmarks and, thus, do not cover the new aspects and characteristics of big data and big data systems. Several proposals have been put forward to modify TPC-DS to cover big data usecases, similar to what we have proposed here with **BigBench**. For example, Zhao et al. propose Big DS, which extends the TPC-DS model for social marketing and advertisement applications [23]. However, Big DS is currently in the early stage of design—a data model and query set are not available. We believe that once the benchmark has been better defined, it would be possible to complement BigBench with the extensions proposed by Big DS. Another TPC-DS variant is proposed by Yi and Dai, as part of the HiBench ETL benchmark suite [8]. The authors extend the TPC-DS model to generate web logs, similar to BigBench. Once again, we believe that the specific extensions could be relatively easily incorporated into BigBench in future. Several other proposals have been made for component benchmarks that test specific functions of big data systems. Notable examples are the Berkeley Big Data Benchmark, the benchmark

presented by Pavlo et al. [13], and BigDataBench, a suite similar to HiBench and mainly targeted at hardware benchmarking [20]. Although interesting and useful, these benchmarks do not present an end-to-end scenario and, thus, have a different focus than BigBench.

In November 2013, the TPC announced the creation of a Big Data Working Group (TPC-BD)[1], which recently released the TPCx-HS benchmark (TPC Express Benchmark for Hadoop Systems) in August 2014[2]. TPCx-HS is based on the TeraSort benchmark, which is a relatively simple Hadoop-based sort benchmark that has been successful in establishing an annual sorting competition[3].

Additionally, there are other active efforts in the database community as well as the high-performance computing community in the area of graph benchmarks. A well-known graph benchmark is the Graph 500, developed by the HPC community [11]. Official benchmark results are published in the Graph 500 list[4]. Another example is LinkBench [1], a benchmark that models the social graph of a social application. A general discussion of graph database benchmarks can be found in [6].

3 BigBench Overview

BigBench [7] is an end-to-end big data benchmark based on TPC-DS [15], TPC's latest decision support benchmark. TPC-DS is designed with a multiple-snowflake schema populated with structured data allowing the exercise of all aspects of commercial decision support systems, built with a modern database management system. The snowflake schema is designed using a retail model consisting of three sales channels, *Store, Web* and *Catalog*, plus an *Inventory* fact table. BigBench's schema uses the data of the Store and Web sales distribution channels of TPC-DS and augments it with semi-structured and unstructured data.

The semi-structured part captures registered and guest user clicks on the retailer's website. Some of these clicks are for completing a customer order. As shown in Fig. 1, the semi-structured data is logically related to the *Web Page, Customer* and *Sales* tables in the structured part. The design assumes the semi-structured data to be a key-value format, similar to Apache web server log format. Typically, database and MapReduce (MR) systems would convert this format to a table with the following five columns (DateID, TimeID, SalesID, WebPageID, UserID). However, such conversion is not necessary, since some systems may choose to run analytics on the native key-value format itself.

Product reviews—a growing source of data in online retail sales—is used to populate the unstructured part of the BigBench data model. Figure 1 shows product reviews on the right-hand side, and its relationship to Item, Sales, and Customer tables in the structured part. A possible implementation for the

[1] www.tpc.org/tpcbd/.
[2] www.tpc.org/information/other/tpcx-hs%20press%20release_final.pdf.
[3] http://sortbenchmark.org.
[4] http://www.graph500.org/.

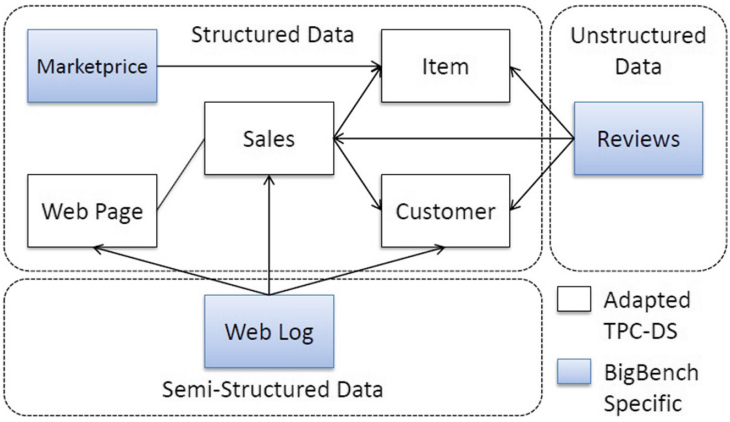

Fig. 1. BigBench logical data schema

product reviews data is via a single table with the structure: (DateID, TimeID, SalesID, ItemID, ReviewRating, ReviewText).

BigBench employs a data generator that is based on PDGF [17], a parallel data generator capable of producing large amounts of data in a scalable and high performance fashion. PDGF "plugins", which are java extensions, enable the program to generate data for any arbitrary schema. Using such plugins, PDGF can generate data for all three parts of the BigBench schema, viz., structured, semi-structured and unstructured. The weblogs, representing the semi-structured part of the schema, are generated using a key-value plugin. Product reviews (the unstructured part) are generated using a Markov Chain plugin. The algorithm produces synthetic text by extracting key words from sample input into a dictionary and applying Markov Chain techniques to generate arbitrary text. Sample data was taken from publicly available data at the Amazon website. PDGF has been programmed to generate a BigBench database of any size between 1 GB and 1 PB (petabyte). Some tables, such as Customers, scale sublinearly, to avoid unrealistic table sizes, whereas other tables, e.g. Sales and Returns, scale linearly.

The BigBench query workload includes 30 queries, of which the ten queries that operate only on the structured part of the schema have been taken from the TPC-DS workload. The remaining 20 queries were adapted from a McKinsey report on big data use cases and opportunities [9]. Of those, 7 queries run on the semi-structured part of the schema; 6 queries run on the unstructured part; and the remaining run on the structured part.

Similar to many current big data systems, BigBench employs batch-oriented processing. Following the precedent established by other, similar (TPC) benchmarks, the preferred performance metric is a single, "abstract" value that is used for comparing end-to-end performance of different big data systems. Thus, the proposed metric, which is loosely based on the TPC-DS metric, includes the following [16]:

– T_L: Execution time of the loading process;
– T_P: Execution time of the power test;
– T_{TT1}: Execution time of the first throughput test;
– T_{DM}: Execution time of the data maintenance task.
– T_{TT2}: Execution time of the second throughput test;
– BBQ_pH: BigBench Queries per Hour;

$$BBQpH = \frac{30 * 3 * 3600}{T_L + T_P + \frac{T_{TT1}}{S} + T_{DM} + \frac{T_{TT2}}{S}} \tag{1}$$

$$BBQpH = \frac{30 * 3 * S * 3600}{S * T_L + S * T_P + T_{TT1} + S * T_{DM} + T_{TT2}} \tag{2}$$

4 Experiments

In the experiments reported here, the BigBench workload was executed on two test platforms—a 6-node cluster ("Small") and a 544-node cluster ("Large"). The test dataset was generated using the BigBench data generator described in [7]. The dataset size was selected as 1 TB (i.e. ScaleFactor, SF = 1000). The tables with linear growth rates make up the bulk of the dataset, as explained in [16]. All the dataset tables were created in Hive.

Benchmark results were produced using the implementation of BigBench for the Hadoop ecosystem described in [3]. The implementation uses four open-source software frameworks: Apache Hadoop, Apache Hive, Apache Mahout, and the Natural Language Processing Toolkit (NLTK). These frameworks are used to implement the 30 queries employing one of the following methods:

– Pure Hive, for queries 5, 6, 7, 9, 11, 12, 13, 14, 17, 21, 22, 23, 24
– Hive with MapReduce programs, for queries 1, 2
– Hive with Hadoop streaming, for queries 3, 4, 29, 30
– Apache Mahout, for queries 15, 20, 25, 26, 28
– Apache OpenNLP, for queries 10, 16, 18, 19, 27

4.1 Test Platforms

The two clusters used for testing represent two distinct points in the scale-up spectrum of Hadoop clusters. The "Small" cluster had 6 dual-socket servers, while the "Large" cluster had 544 dual-socket servers. Details of the cluster configurations are shown in Table 1. The large cluster results are from the Pivotal Analytics Workbench[5], made available by Pivotal Software, Inc. The benchmarking effort on that platform was supported by a grant from Pivotal to the Center for Large-Scale Data Systems Research (CLDS) at the San Diego Supercomputer Center, UC San Diego.

[5] http://www.analyticsworkbench.com.

Table 1. Configuration of test clusters

Cluster configuration	Small	Large
Processor per node	2 × Xeon E5-2680 v2 @2.80 GHz	2 × Xeon X5670 @2.93 GHz
Core/Thread per node	20/40	12/24
Main Memory per node	128 GB	48 GB
Storage per node	12 × 2 TB HDD 7.2 Krpm	12 × 2 TB HDD 7.2 Krpm
Total HDFS storage	90 TB	9,420 TB
Cluster interconnect	10 Gb ethernet	10 Gb infiniband
OS type	CentOS 6.5	RHEL 6.1 64-bit
Hadoop version	Cloudera CDH5	Pivotal HD 2.0.1
JDK version	1.7	1.7
Name node	1	1
Data node/Tasker node	4	542
Hive server	1	1

4.2 Experimental Observations

The 30 BigBench queries were run sequentially on each test cluster and statistics were collected for each query. The results presented here are from running the queries without any prior tuning of the systems. Thus, these results represent the "raw, out-of-the-box" performance of each system. While the performance of a number of the queries could improve significantly with careful tuning, the analysis of data collected in this initial set of tests nonetheless provides useful insights into the general characteristics of the workload and, thus, into the applicability of the benchmark itself.

The first step of the experiment consists of loading the SF = 1000, 1 TB dataset into the Hive tables. On the large cluster this operation took almost twice as long as on the small cluster (87 min vs. 48 min). This behavior is the first indication that the 1 TB database, while appropriate for the small cluster with 4 data nodes, is highly undersized for the large cluster with 544 data nodes. Staging and replicating a relatively small amount of data over a large number of Hive data nodes results in overheads that dominates the performance of the data ingestion process.

In the next step of the experiment, the queries were run sequentially, and the execution time was collected for each query. Table 2 presents the query execution times as measured on both the small and large clusters.

Comparing the query execution times between the two clusters highlights the lack of tuning prior to query execution as well as the over-scaling of the large cluster, given that the data set is relatively small for a cluster of that size. Some queries are highly parallelizable and are, thus, able to take advantage of the significantly more resources available in the large cluster in order to perform queries much faster than on the small cluster. However, a number of queries perform slower on the large cluster due to the under-scaling of the data set as well as lack of tuning.

Table 2. Query execution times for small and large clusters

Query	Small(min)	Large(min)	Query	Small(min)	Large(min)
1	5.9	3.6	16	11.7	3.8
2	11.4	3.7	17	3.9	5.7
3	9.8	4.0	18	11.7	10.0
4	908.1	28.8	19	6.2	7.0
5	177.0	16.5	20	14.7	6.0
6	9.7	4.9	21	7.3	3.8
7	14.0	9.9	22	31.9	7.1
8	29.6	10.9	23	107.5	39.8
9	8.0	4.0	24	5.8	3.7
10	10.1	13.4	25	5.5	3.9
11	2.4	2.0	26	7.1	4.1
12	5.1	9.3	27	0.6	0.8
13	5.4	6.6	28	1.9	19.6
14	2.5	1.7	29	24.3	3.6
15	5.1	1.4	30	44.7	6.7

Additional insight can be gained by examining the system utilization statistics that were collected during the experiment. Two queries that were run on the small cluster are presented here to illustrate the two main cases that were observed. In the first, the query is able to take advantage of the system resources provided without the need for tuning, as is the case for query Q16. As shown in Fig. 2, the resource utilization is well balanced throughout the execution of the query. Demand for CPU resources spans the entire query execution period. Similarly, the disk activity is also distributed across the duration of the query, and not localized to a small subset of the query execution time. Memory utilization is also relatively uniform over the execution time, while staying at a comfortable distance from saturation. Lastly, inter-node communication shows two strong bursts of activity, which is likely driven by the *map* and the *reduce* steps.

In contrast, in the second case, the query has a very skewed profile for system resource usage. This is exemplified in Q1, as shown in Fig. 2. The resource utilization of the query is characterized by a burst of CPU and disk activity at the very beginning, followed by a very low level of activity for the remainder of the query execution time. This is associated with a poor usage of available memory resources followed by a final burst of network communication toward the very end of the query execution. Much work remains to be done to fully characterize the behavior of these un-optimized queries. It is likely that the query uses the default number of mappers set by Hive and could benefit from a much large number of tasks (Fig. 3).

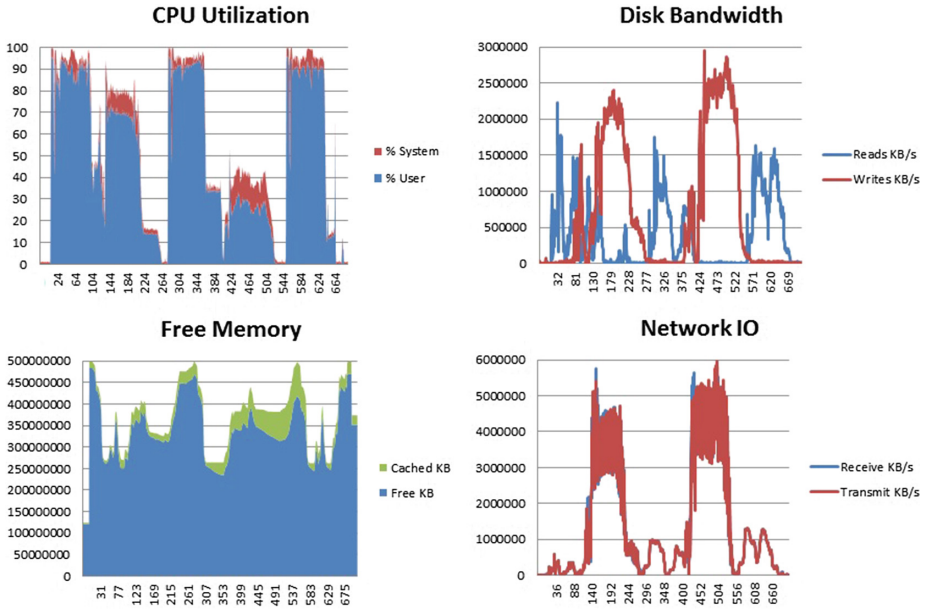

Fig. 2. System utilization statistics for Q16

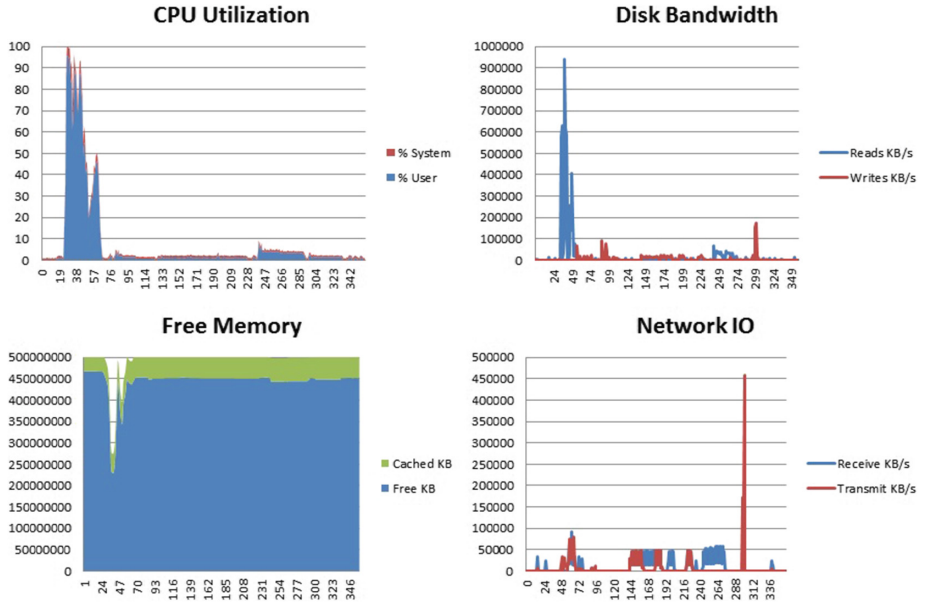

Fig. 3. System utilization statistics for Q1

Through this initial set of experiments, we were able to confirm that the BigBench queries represent a solid challenge for Hadoop clusters of different sizes. The query set displayed a wide spectrum of behaviors that necessitate careful tuning before reaching a balanced utilization of all major system resources. Furthermore, during the experiments we also noted that the benchmark queries could be used for component testing. To focus the testing on a selected cluster component, one can run specific queries that apply particular stress patterns on given components, without having to run the entire suite of queries. However, unlike micro-benchmarks, these focused tests are directly related to specific use-cases as highlighted by the business description that the benchmark provides for each query.

In these experiments, the small versus large clusters also represent different execution environments. The small cluster consists of a limited number of nodes, which are all dedicated to this task. Whereas, the large cluster consists of a few hundreds multi-tenancy nodes. While the 544 nodes that were used were dedicated to this experiment, they were part of a larger cluster of 1000 nodes that was shared with other applications running on the other nodes.

In this benchmark experiment, we also took the approach of running in "Power" mode, where each query is executed individually in "stand-alone" mode, leading to a better understanding of its performance behavior. However, the benchmark is also designed to run in the so-called "Throughput mode", where multiple parallel streams of queries can run concurrently. The benchmark provides a single metric that combines results from both these modes of execution—*Power* mode and *Throughput* mode, in order to provide a simpler metric that can be used for comparison.

5 Technical Discussion of the Workload

In this section, we discuss the technical aspects of the 30 BigBench queries. The discussion is separated in two parts: a description of the generic characteristics of the workload, followed by details of a Hive-specific implementation.

5.1 Generic Workload Characteristics

As mentioned in Sect. 3, the workload dataset can be separated into three categories: structured, unstructured, and semi-structured data. BigBench inherits the general scaling properties of TPC-DS, however, unlike TPC-DS it does not restrict scaling to discrete, predefined scale factors. Instead, it provides for a continous scaling model. The database size can range from 1 GB to 1 PB. Linearly scaled tables, e.g. the "fact" tables, will have about 1,000,000 times more records for the 1 PB data set than for the 1 GB data set. Other tables, e.g. the "dimension" tables, such as, *Customer* or *Store*, use logarithmic or square root scaling. As a result, query input sizes are not necessarily linearly dependent on the scaling factor. This can be seen in Table 3, where the difference of query input sizes for Scale Factor SF = 1 is only 7.5 (57 MB : 479 MB), whereas it is

Table 3. Input and output of the 30 queries

Query	# Tables	Input size (SF 1/ SF 1000)	Query	# Tables	Input size
1	2	59 MB/69 GB	16	5	100 MB/103 GB
2	1	88 MB/122 GB	17	7	92 MB/70 GB
3	1	88 MB/122 GB	18	3	112 MB/71 GB
4	4	109 MB/122 GB	19	5	83 MB/9 GB
5	4	180 MB/123 GB	20	2	57 MB/72 GB
6	4	159 MB/168 GB	21	6	154 MB/171 GB
7	5	87 MB/70 GB	22	5	429 MB/70 GB
8	4	165 MB/221 GB	23	4	429 MB/70 GB
9	5	148 MB/69 GB	24	4	86 MB/99 GB
10	1	58 MB/2 GB	25	2	131 MB/168 GB
11	2	135 MB/101 GB	26	2	59 MB/69 GB
12	3	147 MB/122 GB	27	1	58 MB/2 GB
13	4	159 MB/168 GB	28	1	58 MB/2 GB
14	5	83 MB/99 GB	29	2	82 MB/99 GB
15	2	59 MB/69 GB	30	2	93 MB/122 GB

111 for SF = 1000 (2 GB : 221 GB). The table shows the number of tables as well as the input sizes for each query.

Out of the 30 queries, seven reference semi-structured data, six reference unstructured data, while 17 queries reference the structured part of the data.

5.2 Workload Characteristics of the Hive Implementation

The Hadoop-based implementation uses a range of programming techniques to implement the different queries. The workload consists of MapReduce jobs, HiveQL queries, Hadoop streaming jobs, Mahout programs, and OpenNLP programs. For the Hadoop streaming jobs, multiple implementation strategies are used, including command line programs, Java programs, and Python programs. The Mahout jobs are executed outside of Hive, unlike all other parts of the workload. OpenNLP programs are integrated into HiveQL as user defined functions (UDFs). In Table 4, an overview of which type of query uses which type of processing model can be seen.

As shown in the table, 14 out of 30 queries are pure HiveQL queries. Four queries are implemented using Python, two are Java-based MR jobs. Five queries use the OpenNLP libraries to implement sentiment analysis and named-entity recognition. And, finally, five queries use Mahout to implement machine learning algorithms. It should be noted that all jobs use Hive as a driver, and also for data processing.

Table 4. Query implementation techniques

Query	Processing model	Query	Processing model
1	Java MR	16	OpenNLP sentiment analysis
2	Java MR	17	HiveQL
3	Python streaming MR	18	OpenNLP sentiment analysis
4	Python streaming MR	19	OpenNLP sentiment analysis
5	HiveQL	20	Mahout k-means
6	HiveQL	21	HiveQL
7	HiveQL	22	HiveQL
8	HiveQL	23	HiveQL
9	HiveQL	24	HiveQL
10	OpenNLP sentiment analysis	25	Mahout K-means
11	HiveQL	26	Mahout K-means
12	HiveQL	27	OpenNLP named-entity recognition
13	HiveQL	28	Mahout naive bayes
14	HiveQL	29	Python streaming MR
15	Mahout K-Means	30	Python streaming MR

6 Community Feedback

In this section, we summarize the feedback received from a number of sources including the organizations represented by the authors; some of the customers of some of these organizations; and, from direct interviews with several individuals representing the Hadoop community at large. In addition to the typical issues involved in creating a new benchmark, defining a benchmark for big data applications is particularly challenging due to evolving nature of this new field. The key takeaway from the feedback received is the tension between the desire to extend the BigBench specification to cover many more use cases and technology stacks, versus the requirement to keep the benchmark simple and compact for ease of use and comparison. We explore how we plan to balance this trade-off and prioritize the evolution of our benchmark in the upcoming Sect. 7.

Positive feedback. A significant portion of the feedback we obtained expressed appreciation for the effort to create such benchmark, and for many of the technical choices we made. There was positive consensus around the choice of starting from a known benchmark, such as TPC-DS. The community's familiarity with that benchmark and the fact that available TPC-DS implementations could serve as partial implementations of BigBench, were viewed as a clear plus. Also, there was agreement that a relational-only benchmark does not capture key aspects of real-life usecases. Thus, the non-relational extensions that were presented were well received. Providing a reference implementation was also highly appreciated.

While there were some suggestions regarding the specific details of the implementation, most interviewees agreed with the approach and the basic choices that were made.

Common misunderstandings. While having a reference implementation is critical to fostering adoption, we also realized that this makes it easy to misconstrue the benchmark as being prescriptive about a specific combination of frameworks that happened to be chosen for the implementation, e.g., say, Hive/Hadoop. For example, we heard the following question a number of times: "Is this just a Hive benchmark?", or "Is this just for relational data?". The existence of an implementation biases interpretation of the benchmark goals, to the extent that more than one individual missed the fact that the benchmark specification, and the implementation, contain several non-relational components. We expect that this will become less problematic as the benchmark gains traction and different implementations start to emerge that use other frameworks. For the time being, we will address such questions by simply providing a clear description of the scope and goals of the benchmark, and emphasize that the current implementation is a reference implementation, and not mandatory.

Technology coverage. A common set of requests were about adding features to the benchmark that stress a specific technology:

1. *Graph Analytics* is probably one of the number one asks we hear form the community. Different sources reported that the ability to ingest, update, analyze large graphs is an important technological challenge faced by organizations todays. For example Jakob Homan from LinkedIn remarked: "There are the big players like FB, LI and Twitter, but pretty much every organization has some type of graph that it uses to drive engagement."
2. *Streaming* is the second most cited ask for our benchmark. The ability to process a continuous feed of data (e.g., tweets, user posts, server logs), and perform filtering, projection, aggregations, trend detection, outlier detection, etc. in a near real-time fashion, seems to be another key scenario people consider a big data problem. Thomas Graves from Yahoo! for example ask us to consider Storm [10] and Spark [22] to extend our current benchmark to capture streaming use cases.
3. *Interactive Querying.* The support for fast ad-hoc queries on top of a large set of data was another technology stack considered. The argument was towards supporting the large number of small interactive operations performed by data scientist while exploring a data set and devising new analysis/algorithms.

Beside the specific technology, people expressed strong feelings about having a benchmark capable of capturing the following two aspects:

1. *Multi-tenancy:* speaking with large cluster operators, they strongly underlined the need to exercise the multi-tenancy capabilities of a big data stack. Often benchmarks are focused on latency/throughput for a single run of workload performed in a dedicated set of machines. This often allows for over-tuning

of the execution environment to perfectly serve a single run, making the benchmark too synthetic, and more generally does not match the typically operating conditions of the systems under test.

2. *Fault-tolerance:* another key concern for big data developers and cluster operators is fault-tolerance. At the typical scale of big data systems, the sheer volume of hardware/software components involved makes "faults" a common condition. Capturing this in the benchmark seems to be an important requirement. There are two key dimensions to this problem: a functional aspect, e.g., no data are lost despite faults, and performance one, e.g., graceful degradation of throughput and latency under faulty conditions. Moreover capturing "limping" hardware beside all-or-nothing faults seem an interesting extra dimension.

Use case coverage. A final set of concerns was related to the choice of a specific vertical use-case. The concern being that the specifics of the use case we picked was potentially skewing the attention towards certain functionalities more than other. Concretely this was spelled out as a request to broaden the spectrum of use cases considered, particularly to include advertisement and social-network scenarios.

Limiting Complexity. Most of the above comments are pushing us towards making our benchmark richer and broader. This is balanced by the need, express implicitly or explicitly by multiple interviewee, to maintain the size and complexity of the workload contained. Providing a reference implementation allow users to bare significantly more complexity, but the onerous cost of porting this benchmark to an alternative technology stack grows dramatically with the complexity of the benchmark. Moreover, a benchmark that is too complex and faceted makes interpretation and comparison of the results very problematic, reducing the value of the benchmark as a tool to compare solutions.

In the following section, we address the above comments, and propose an agenda on how to extend the benchmark accordingly.

7 Extending BigBench

BigBench is an end-to-end benchmark that focuses on structured data and declarative workloads with additional support for unstructured data and procedural workloads. This section highlights several possible extensions to BigBench that can potentially make the benchmark more representative of a broader variety of real-life big data workloads.

Incorporating Concurrency. The benchmark defines a model for submitting concurrent workload streams in parallel and for randomizing the workload across the multiple query streams [16]. This is intended to cover multi-tenant scenarios where multiple instances of the same workload or single instances of multiple workloads could execute in parallel. Example of a concurrent/complex workload

w composed of two elemental workloads w_1 and w_2 could be: $w = n_1 * w_1 + n_2 * w_2$, where n_1 and n_2 are the number of instances of w_1 and w_2 respectively. The query concurrency models in several existing online transactional processing (OLTP) and online analytical processing (OLAP) industry standard benchmarks serve as a good starting point [14,15,19].

Improving Procedural Coverage. BigBench has two procedural workloads defined at the moment: K-Means and Bayes. Both are representative of the machine learning domain and their respective specifications define a dependency on a relational database or suchlike. BigBench could be extended to include "pure" procedural workloads that process unstructured data without requiring format conversion. These workloads would also represent categories that are somewhat under-represented in BigBench, including web-based and component-level benchmarks. *PageRank* is a good representative of web-based workloads, while *WordCount, SleepJob* and *Sort* are excellent representatives of component level benchmarks.

Including Other Metrics. The specification and reference implementation should be extended to measure other metrics important to technology choices, such as price/performance, energy efficiency, and performance under failures. Price/performance and energy efficiency are already included in various industry standard benchmarks. Performance under failures is an important consideration for big data systems, which run on large scale-clusters, and consequently, partial component failures such as hardware failures can be common.

Incorporating Incremental Data Uploads. In real-world deployments, big data applications ingest data incrementally, rather than re-loading the entire dataset. For example, tables are typically implemented as a collection of time-based partitions to support data refresh. Each partition stores data for a time slice, e.g., one hour or one day. Whenever new data arrive, they are loaded as new partitions, or aggregated with an existing partitions to create a new partition. Thus, there never a need to reload the entire data. In the future, Bigbench could account for such partition-based data refresh strategies.

Incorporating Additional Workloads. TPC-DS is designed to evaluate the performance of decision-support style queries of data warehouse systems. However, constrainedonly OLAP queries. Many real-world big data systems, also encounter periodic workloads, i.e. workloads that repeat hourly, daily, or even weekly, which are different from OLAP queries. A possible extension to BigBench is to include such kind of workloads to better simulate the real-world Big Data systems. Some good candidates of such workloads include the off-line collaborative filtering analysis of all items [21], unstructured data indexing and ranking for intranet search service, user authority or similarity analysis, etc.

8 Towards an Industry Standard Benchmark

As with the development of any software product, the process of turning a benchmark idea into a product is not trivial. The three most recognized industry standard consortia, namely the Standard Performance Evaluation Corporation (SPEC), the Transaction Processing Performance Council (TPC) and the Storage Performance Council (SPC) have developed processes to organize benchmark development; deal with benchmark evolution, i.e., versioning; and publish benchmark results to ensure successful benchmarking. The TPC, has managed to retain continuity of benchmarks over a few decades, while keeping the benchmarks comparable. This has provided companies the ability to compare benchmark results over a very long time period and across many products. In this section, we describe the necessary steps and discuss the advantages and disadvantages of developing an industry specification that is similar to TPC.

All TPC benchmark specifications developed so far have been technology agnostic, i.e., they specify a workload without using terms of any particular architecture or implementation by defining a set of functional requirements that can be run on any system, regardless of hardware, database management software or operating system. Furthermore, they follow a similar methodology and, consequently, follow a similar structure. It is the responsibility of those measuring the performance of systems using TPC benchmarks, a.k.a. the test sponsor, to implement their setup compliant with the benchmark specification and to submit proof that it meets all benchmark requirements, i.e., that the implementation complies with the specification. The proof has to be submitted with every benchmark publication in form of a full disclosure report. The intent of the full disclosure report is to enable other parties to reproduce the performance measurement. This methodology allows any vendor, using "proprietary" or "open" systems, to implement TPC benchmarks while still guaranteeing end-users that the measurement is comparable.

The above approach to benchmarking broadens the applicability of benchmark specifications to many architecture and allows for the optimal implementation of a specific product on a specific platform. At the same time it makes the first benchmark publication very costly, often too costly, because any new implementation needs to be reviewed by an independent auditor. As a consequence the TPC has started to develop a novel way to specify benchmarks. The new benchmark category is labeled TPC Express so that it can easily be distinguished from the traditional category, which is labeled TPC Enterprise. TPC Express benchmarks are based on predefined, executable benchmark kits that can be rapidly deployed and measured. Providing a benchmark kit focuses on a critical subset of system, trading the ability to demonstrate absolute optimal performance for improved ease and costs of benchmarking (Table 5).

Summarizing the differences between enterprise and express benchmark specifications, it seems that enterprise benchmark have a higher price tag, and are more time consuming compared to express benchmarks. However their implementation is limited to the technology that is supported in the KIT.

Table 5. Comparison enterprise and express benchmark models

Enterprise	Express
Specification-based with tools provided by the TPC to build the data sets and workloads	Kit-based that runs the benchmark end-to-end, including tools provided by the TPC to build data sets and workloads
Benchmark publication specific implementation, i.e. each benchmark publication can be different	Out of the box implementation, i.e. each benchmark publication follows the same implementation
Best possible optimization allowed	System tuning for "unalterable" benchmark application
Complete Audit by an independent third party	Mostly self validation augmented by peer-reviews
Price required	Price eliminated
If Atomicity, Consistency, Isolation and Durability (ACID) are required as part of the benchmark, full ACID testing needs to be done as part of any benchmark publication	If ACID is required as part of the benchmark, ACI testing is conducted as a part of self validation. Durability cannot be tested as it requires an auditor to assure correctness
Large variety of configurations	Limited number of configurations focused on stressing key components of the benchmark
TPC revenues from benchmark registration	TPC revenues from license sales and potentially also benchmark registration
Substantial implementation costs	Reduced implementation costs
Ability to promote results as soon as published to the TPC	Ability to promote results as soon as published to the TPC

The express benchmark model is very promising as it will lower the entry cost into benchmarking as well as per benchmark publication costs. The big hurdle for express benchmarks is the development of a KIT. BigBench defines queries using functional specifications [2] allowing BigBench to accommodate the diverse and rapidly evolving nature of big data technologies (e.g., MapReduce, Hive, Spark, etc.). Currently, BigBench includes a Hive-based reference implementation. The intent is that for each query there could be multiple implementations satisfying the benchmark's functional specification. To increase rapid adoption of the benchmark, it would be beneficial to make all valid implementations available as open source to a central repository. The resulting repository can be used to aid a BigBench express KIT.

The specification will be extended to provide implementation guidelines to ensure that the essential big data principles are maintained. For example, all file formats used in an implementation must demonstrate the expected flexibility of

being able to be created, read, and written from multiple popular engines on the Hadoop stack, e.g., (MapReduce, Pig, Hive). Such formats ensure that all data is immediately query-able, with no delays for ETL. Costly data format conversion is unnecessary and thus no overhead is incurred.

In addition to having a KIT, for a possible TPC big data express benchmark one will need to develop the following sections:

- Introduction/Preamble. This section includes a high level introduction to the benchmark and general implementation guidelines. The implementation guidelines if adopted from the TPC exists as a boilerplate in every benchmark, and can be used with minor modifications. However, special implementation guidelines can be easily incorporated. For instance, in order to give multiple popular engines access to the data without incurring costly data conversion overhead, it might be beneficial to provide guidelines in the BigBench specifications to ensure that the data formats used in benchmark implementations ensure that essential big data principles are maintained. For example, all file formats used in an implementation must demonstrate the expected flexibility of being able to be created, read, and written from multiple popular engines on the Hadoop stack, e.g., (MapReduce, Pig, Hive).
- Data/Database Design: Requirements and restrictions on how to implement the database schema. In case of the express model this section can be relatively short as only modifications to the KIT need to be discussed. Otherwise the KIT is what needs to be run.
- Workload Scaling: Tools and methodology on how to scale the workload. This would include a description and usage of the tools plus methods to scale the data and potentially the workload.
- Metric and Execution Rules: Again the KIT will serve as a reference implementation of the metric and execution rules. This section only needs to description, on a high level, how to execute the benchmark and how to derive metrics. Additionally, it needs to describe any deviations allowed from the execution implemented in the KIT. This section would also include extensions to BigBench to measure other metrics important to technology choices, such as performance-per-cost, energy efficiency, and performance subject to failures. Performance-per-cost and energy efficiency are already included in various industry standard benchmarks. Performance subject to failures is an important metric as big data technologies run on large scale clusters, and consequently, partial component failures such as hardware failures can be common.
- Pricing: This section will cover pricing related wording specific to BigBench. Generic pricing rules are already available TPC's pricing specification.
- Full Disclosure Report (FDR): Every TPC benchmark publication includes an FDR that allows anybody to reproduce the benchmark. In case of an express benchmark only allowed deviations from the KIT and system specifics need to be included in the FDR and, hence, the specification wording is limited to that.
- Audit Requirements: Minimum requirements for the audit process that need to be followed. In case of an express benchmark, self auditing scripts that show correct implementation and execution of the benchmark need to be included and, if desired, rules for peer-auditing.

9 Conclusion

As big data analytics becomes an important part of todays data management ecosystem, there is a need for an industry standard benchmark that can measure the performance and price-performance aspects total system under realistic workloads. In this paper, we propose a framework for an end to end big data analytics benchmark based on BigBench. The benchmark is intended to represent todays data management ecosystem which is implemented as an extension of enterprise DW application (structured data) with new data sources (semi-structured and unstructured). The paper presents 30 queries representative of real life scenarios, their characteristics and experiment results. This paper is presented as a proposal to the TPC to create the next generation industry standard benchmark that can be developed as an Express benchmark or Enterprise benchmark.

BigBench currently incorporates a retail industry use case. Recent customer surveys reveal additional important and common use cases from other industries, e.g., the financial industry [4]. Hence, as additional surveys and empirical data emerge, BigBench will be extended to incorporate additional use cases.

Acknowledgements. Portions of the research in this paper use results obtained from the Pivotal Analytics Workbench, made available by Pivotal Software, Inc. Work performed by co-authors Baru and Youn was partially supported via industry sponsorship from Pivotal and Intel of the Center for Large Scale Data Systems Research (CLDS) at the San Diego Supercomputer Center, UC San Diego and by a grant from the Information Technology Laboratory (ITL) of the National Institute for Standards and Technology (NIST).

References

1. Armstrong, T.G., Ponnekanti, V., Borthakur, D., Callaghan, M.: LinkBench: a database benchmark based on the facebook social graph. In: SIGMOD, pp. 1185–1196 (2013)
2. Chen, Y., Raab, F., Katz, R.: From TPC-C to big data benchmarks: a functional workload model. In: Rabl, T., Poess, M., Baru, C., Jacobsen, H.-A. (eds.) WBDB 2012. LNCS, vol. 8163, pp. 28–43. Springer, Heidelberg (2014)
3. Chowdhury, B., Rabl, T., Saadatpanah, P., Du, J., Jacobsen, H.A.: A BigBench implementation in the hadoop ecosystem. In: Rabl, T., Raghunath, N., Poess, M., Bhandarkar, M., Jacobsen, H.-A., Baru, C. (eds.) WBDB 2013. LNCS, vol. 8585, pp. 3–18. Springer, Switzerland (2014)
4. Costley, J., Lankford, P.: Big Data Cases in Banking and Securities - A Report from the Front Lines. Technical report STAC (2014)
5. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
6. Dominguez-Sal, D., Martinez-Bazan, N., Muntes-Mulero, V., Baleta, P., Larriba-Pey, J.L.: A Discussion on the Design of Graph Database Benchmarks. In: Nambiar, R., Poess, M. (eds.) TPCTC 2010. LNCS, vol. 6417, pp. 25–40. Springer, Heidelberg (2011)

7. Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A., Jacobsen., H.A.: BigBench: towards an industry standard benchmark for big data analytics. In: SIGMOD (2013)
8. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: ICDEW (2010)
9. Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Byers, A.H.: Big data: the next frontier for innovation, competition, and productivity. Technical report, McKinsey Global Institute (2011). http://www.mckinsey.com/insights/mgi/research/technology_and_innovation/big_data_the_next_frontier_for_innovation
10. Marz, N.: Storm - Distributed and Fault-Tolerant Realtime Computation. http://www.storm-project.net/
11. Murphy, R.C., Wheeler, K.B., Barrett, B.W., Ang, J.A.: Introducing the Graph 500. Cray Users Group (CUG) (2010)
12. Nambiar, R.O., Poess, M.: The making of TPC-DS. In: Dayal, U., Whang, K.Y., Lomet, D.B., Alonso, G., Lohman, G.M., Kersten, M.L., Cha, S.K., Kim, Y.K. (eds.) VLDB, pp. 1049–1058. ACM (2006)
13. Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., Stonebraker, M.: A comparison of approaches to large-scale data analysis. In: SIGMOD, pp. 165–178 (2009)
14. Pöss, M., Floyd, C.: New TPC benchmarks for decision support and web commerce. SIGMOD Rec. **29**(4), 64–71 (2000)
15. Pöss, M., Nambiar, R.O., Walrath, D.: Why you should run TPC-DS: a workload analysis. In: VLDB, pp. 1138–1149 (2007)
16. Rabl, T., Frank, M., Danisch, M., Gowda, B., Jacobsen, H.A.: Towards a complete BigBench implementation. In: WBDB (2014). (in print)
17. Rabl, T., Frank, M., Sergieh, H.M., Kosch, H.: A data generator for cloud-scale benchmarking. In: Nambiar, R., Poess, M. (eds.) TPCTC 2010. LNCS, vol. 6417, pp. 41–56. Springer, Heidelberg (2011)
18. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: a warehousing solution over a map-reduce framework. PVLDB **2**(2), 1626–1629 (2009)
19. Transaction Processing Performance Council: TPC Benchmark C - Standard Specification (2010). (version 5.11)
20. Wang, L., Zhan, J., Luo, C., Zhu, Y., Yang, Q., He, Y., Gao, W., Jia, Z., Shi, Y., Zhang, S., Zhen, C., Lu, G., Zhan, K., Li, X., Qiu, B.: BigDataBench: a big data benchmark suite from internet services. In: HPCA (2014)
21. Yi, L., Dai, J.: Experience from hadoop benchmarking with HiBench: from micro-benchmarks toward end-to-end pipelines. In: Rabl, T., Raghunath, N., Poess, M., Bhandarkar, M., Jacobsen, H.-A., Baru, C. (eds.) WBDB 2013. LNCS, vol. 8585, pp. 43–48. Springer, Switzerland (2014)
22. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: NSDI, pp. 2–2 (2012)
23. Zhao, J.M., Wang, W., Liu, X.: Big data benchmark - big DS. In: Rabl, T., Raghunath, N., Poess, M., Bhandarkar, M., Jacobsen, H.-A., Baru, C. (eds.) WBDB 2013. LNCS, vol. 8585, pp. 49–57. Springer, Switzerland (2014)