From BigBench to TPCx-BB: Standardization of a Big Data Benchmark

Paul Cao[1], Bhaskar Gowda[2], Seetha Lakshmi[3], Chinmayi Narasimhadevara[4], Patrick Nguyen[5], John Poelman[6], Meikel Poess[7], Tilmann Rabl[8,9]

[1]Hewlett Packard Enterprise, [2]Intel Corporation, [3]Actian Corporation, [4]Cisco Systems Inc., [5]Microsoft Corporation, [6]IBM, [7]Oracle Corporation, [8]Technische Universität Berlin, [9]DFKI GmbH

Abstract. With the increased adoption of Hadoop-based big data systems for the analysis of large volume and variety of data, an effective and common benchmark for big data deployments is needed. There have been a number of proposals from industry and academia to address this challenge. While most either have basic workloads (e.g. word counting), or port existing benchmarks to big data systems (e.g.TPC-H or TPC-DS), some are specifically designed for big data challenges. The most comprehensive proposal among these is the BigBench benchmark, recently standardized by the Transaction Processing Performance Council as TPCx-BB. In this paper, we discuss the progress made since the original BigBench proposal to the standardized TPCx-BB. In addition, we will share the thought process went into creating the specification, challenges in navigating the uncharted territories of a complex benchmark for a fast moving technology domain, and analyze the functionality of the benchmark suite on different Hadoop- and non-Hadoop-based big data engines. We will provide insights on the first official result of TPCx-BB and finally discuss, in brief, other relevant and fast growing big data analytic use cases to be addressed in future big data benchmarks.

1 Introduction

Organizations are increasingly beginning to value big data analytics for improving business, reducing the risks, and solving business challenges. At the same time, they are faced with a number of big data technology and solution options such as: MapReduce, Spark, NoSQL databases, SQL on Hadoop databases, and Flink. Choosing the right technology (or set of technologies) is critical for their success. A standardized benchmark that can be used to evaluate the performance of different big data technologies can greatly help organizations choose the right solution.
Influenced by Moore's law, the rapidly evolving computing and storage landscape enables companies to analyze their data for half the cost every two years. Many companies hope to improve their business model by collecting increasing amounts of data and employing techniques related to big data. Although traditional database systems provide means to store large amounts of data, these have to generally need be in a structured format. In recent years, a large ecosystem of big data tools has evolved, which is

targeted at analyzing the growing amounts of data, structured, semi-structured, or unstructured.
While database systems are well established and their performance is understood by companies, there is no easy methodology to compare, the plethora of big data systems with their many interfaces, APIs, and query languages. In certain situations a scalable big data system can be outperformed by a laptop for real problem sizes [1], emphasizing the need to improve efficiency of scalable big data systems.
Trying to keep up with this rapidly moving trend, customers have the difficult task on their hands to compare cross-platform solutions in order to select the right hardware and software for their big data needs. They rely on industry standard benchmarks to educate, inform and guide making these decisions. An absence of such performance analysis tools in form of standardized benchmarks has magnified customer difficulties, thus motivating the industry to take necessary actions to fill the void.
BigBench [2] was proposed to fill this gap, it set in motion efforts to create an end to end benchmark for big data analytics systems. While it comes with a concrete default implementation, the rules are very flexible regarding the type of systems this work can be run on and how the workloads can be implemented.
Thanks to member companies in the benchmark sub-committee under Transaction Processing Council (TPC), who contributed significant effort in drafting the specification and provide a readily usable benchmark kit, TPCx-BB progressed from being a scientific proposal [2] to an industry standard big data analytics benchmark in a span of two and half years.
In this paper, we describe the process towards a standardized benchmark and show how this process worked for BigBench. In particular, we have the following contributions:

- We give a detailed update of the benchmark and the changes that we required for the standardization.
- We present the first official benchmark submission and give an analysis on the results.
- We give an overview of existing BigBench implementations and compare them based on completeness.

The rest of the paper is structured as follows. In the next section, we give a brief overview of different big data benchmarking proposals. In Section 3, we present TPCx-BB and in Section 4, we describe its standardization process. Section 5 presents TPCx-BB experiments using different big data frameworks. Section 6 gives an outlook on future big data benchmarks and workloads. Section 7 concludes the paper.

2 Related Work

While several benchmarks for big data systems have been proposed, and discussed, most of them are either simplistic (e.g., limited to sorting or counting) or collections of simple use cases rather than end-to-end, application-level benchmarks. While these component benchmarks are good to test individual parts of a big data system, they can-

not provide a holistic view of the performance of the system under test. And more importantly, none of these benchmarks have been discussed and reviewed under the umbrella of benchmark standardization organizations.
The Transaction Processing Performance Council[1] (TPC) understood this need and worked on several benchmarks for the big data space. As a stop-gap solution for MapReduce systems, TeraSort was standardized in TPCx-HS [3]. It is capable of indicating the basic I/O and network throughput of a MapReduce deployment but has limited other information value. Another ongoing work is the revision of TPC-DS [4] for big data systems. To this end, TPC-DS was adapted in Version 2 to accommodate the limitations of current "SQL on Hadoop" systems such has Apache Hive, Apache SparkSQL, and Apache Impala.

3 TPCx-BigBench (TPCx-BB)

Prior workshops on big data benchmarking have concluded that for successful adoption, a benchmark should have some relevance to their use cases, simple to implement, and easy to execute [5]. The TPC has a track record of publishing valuable and widely adopted benchmarks for measuring the performance of database systems. TPC-C, TPC-H, and TPC-DS are noteworthy enterprise benchmarks. Recently the TPC provides another option called as 'TPC Express' standard. Express benchmarks provide ready to run workloads to be executed on specific products. Here workload is bundled in the form of benchmark kits that are ready to run on a number of pre-selected platforms. The express benchmark model is very promising as it will lower the entry cost for test sponsors publishing the benchmark results. However, commitment of resources is required from the kit sponsor to develop, maintain, support and ratify the kit with in the sub-committee, for the lifetime of the kit. In designing the benchmark for big data systems, the TPC applied the lessons distilled from the making of previous successful and not so successful benchmark specifications. For example, with over 250 audited results publications and an even a larger number of publications that had used the benchmark to quantify and demonstrate performance gains from specific HW/SW enhancements, TPC-H is a widely successful benchmark, even though it has been criticized for not being representative of real world decision support workloads at high scale factors. In contrast, there has not been a single audited results published for TPC-DS benchmark, a richer and more comprehensive decision support benchmark, addressing the deficiencies in TPC-H and has been available since 2006. The success and popularity of TPC-H can be attributed to its relative simplicity (8 tables and 22 queries) and timeliness when the database industry was making rapid advances in the data warehousing space and was in need of a relevant benchmark. On the other hand, with TPC-DS, it is a daunting task for end users to comprehend all the 99 queries, the rules for data refresh, the complex business problems designed to model and to analyze their performance. There have been some research publications or competitive analysis using only a subset (or modified versions) of the TPC-DS queries [6].
Balancing the thoroughness of an enterprise benchmark with the flexibility of an express benchmark while keeping the benchmark complexity under check, the TPCx-BB

[1] Transaction Processing Performance Council – www.tpc.org

[7] took a middle of the road approach, in that it limited the number of queries to 30. To keep the benchmark relevant for the big data analytics use cases, the 30 queries are distributed to operate on structured, semi-structured, or unstructured data and using pure HIVE queries, MapReduce, natural language processing, or machine learning libraries. Further, to promote easy and quick adoption of the benchmark, a self-contained kit of the TPCx-BB is made freely available for download from the TPC website[2]. This kit can be used to measure the performance of Hadoop based systems including MapReduce, Apache Hive, and Apache Spark Machine Learning Library (MLlib).

3.1 TPCx-BB Overview

TPCx-BB is a big data batch analytics benchmark inspired by TPC-DS. The benchmark which models aspects of commercial decision support systems for a retail business. TPC-DS consists a snowflake schema representing three sales channels, (store, web, catalog, and online. Each with a sales and a returns table) and inventory fact table. The TPCx-BB uses the store and online distribution channels of TPC-DS and augments it with semi-structured and unstructured data. The prototype proposal of TPCx-BB was been discussed in detail [8].

3.2 Benchmark Kit

The kit is the first application-level benchmark suite specifically designed to measure the performance of big data analytics systems. TPCx-BB measures the performance of Hadoop-based systems including MapReduce, Apache Hive, and Apache Spark and its machine learning library MLlib, and is publicly available for download as a self-contained kit via the TPC Web site.
TPCx-BB's benchmark kit is self-contained to have minimal requirements on external software dependencies and able to run 'out of the box' on the system under test (SUT). The kit is modular and it supports extensibility to new frameworks (i.e. collection of Big Data software/hardware components) can be easily added. The kit consists of three major components as shown in Figure 1, i) the benchmark driver, ii) the workload iii) the data generator.

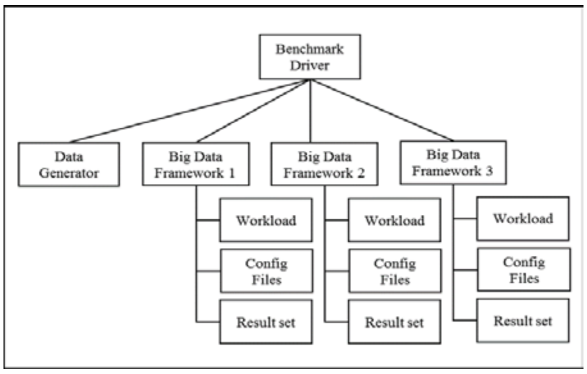

Figure 1 Benchmark Kit

Benchmark Driver. Implemented using Java and Bash scripts, the versatile benchmark driver is the heart of the kit. It orchestrates the workflow involved in executing the benchmark on the SUT. Support for running multiple concurrent query streams, au-

[2] http://www.tpc.org/tpcx-bb

tomated answer set validation, SUT configuration details, and computing the benchmark score are done seamlessly at various phases during the benchmark execution. Additionally, the driver exposes hooks for integrating new frameworks as needed. An advanced mode the benchmark driver provides options to run the complete benchmark or individual queries for testing and optimization purposes.

Data Generator. The kit includes a parallel data generator based on the Parallel Data Generation Framework [9] to generate the input data set required for the benchmark. It is implemented as a Java program that runs as a MapReduce job on the SUT and can generate hundreds of terabytes of data in a relatively short time.

Workload. The kit is designed to have self-contained modules for each framework capable of running the TPCx-BB. All necessary binaries, configuration files, and answer set reside inside the framework module. This makes it easy for kit maintenance and help minimize the impact of adding new frameworks on existing kit modules. Addressing the complexity of big data frameworks and understanding the need to tune and optimize the benchmark, various configuration files provide sufficient hooks to tune the full benchmark or each individual queries by passing run time optimization parameters. Spark machine learning library suite is used for those queries invoking machine learning stages. OpenNLP framework is packaged with the kit for procedural programs invoking natural language processing.

3.3 Supported Big Data Frameworks

Big Data Ecosystem. Big data has transformed industries and research, spawning new solutions for addressing a wide range of technical challenges. Big data ecosystem today offers different end-to-end analytic strategies, scale-up frameworks for operational analytics, and scale-out platforms for advanced analytics.
Scale-up frameworks offer vertically integrated analytical workflows for medium scale big data datasets, e.g. database, data warehousing and online analytical systems. Scale-out frameworks on the other hand offer an array of frameworks closely mimicking high performance computing systems for analytics workflows requiring processing large complex datasets, e.g., MapReduce, Spark.
There are a number of execution frameworks that are part of the Hadoop ecosystem, including MapReduce, Spark, Tez, Flink, Storm, and Samza, each with its own strengths and weaknesses. Initially Hadoop was developed as a special-purpose infrastructure for big data with MapReduce handling massive scalability across hundreds or thousands of servers in a cluster. A number of vendors have developed their own distributions, adding new functionality or improving the code base derived from the Apache open source community. The most popular of these distributions are Cloudera, Hortonworks, MapR and IBM BigInsights each with their unique set of offerings.

SQL on Hadoop. One of the three V's used to describe Big Data is "Variety." Despite the diversity of data stored in Big Data systems, much of it still structured or can be transformed into a form with enough structure that a broad range of useful queries can be expressed in SQL. Evidence that SQL is still popular in the big data space can be seen in the plethora of SQL on Hadoop offerings available today. Some of these SQL engines for big data were built from the ground up to address big data problems, but many have a much longer history. For example, traditional database vendors including Oracle, Teradata and IBM have come out with versions of their SQL engines that run on Hadoop clusters.

One of the earliest and perhaps the most widely known SQL on Hadoop engines is Apache Hive. Hive supports a SQL-like language called HiveQL. Hive can execute queries using MapReduce2, Tez, or Spark. The TPCx-BB kit supports execution of the benchmark using Hive in all three of these frameworks. Besides Hive, there are several other SQL engines in open source, such as Apache Drill, Apache Phoenix, SparkSQL, Cloudera Impala, Teradata Presto, and Pivotal Hawq. Work is being done to have SparkSQL to fully support TPCx-BB, at the time of writing this paper, SparkSQL with help of support patches can successfully run all 30 queries. With the release of Spark 2.0, it is expected TPCx-BB should be able run on SparkSQL with no additional patches.

Non Hadoop Frameworks. TPCx-BB is a good fit for engines designed for processing or aggregating large amounts of data and that can either natively execute the machine learning and natural language processing required by BigBench, or can call out to other engines or frameworks such as Spark.
Since TPCx-BB kit has a pluggable architecture, support for additional SQL engines can be added over time. In fact, any engine capable of answering the 30 BigBench queries is a candidate for inclusion in the kit. The query syntax used by a given engine does not matter, since TPCx-BB allows the 30 use cases to be expressed in any SQL-like query language or natively written programs. However, since the queries are already expressible and available in HiveQL, developing implementations for SQL over Hadoop engines is usually straight forward and less involved than for engines whose query syntax is not similar to SQL. The benchmark prototype was implemented on two non-Hadoop frameworks, namely Apache Flink and Metanautix. As a matter of fact, the first BigBench prototype was actually implemented in Teradata Aster SQLMR.
Apache Flink is a big data streaming dataflow processing engine compatible to the Hadoop stack. It is based on the Stratosphere project [10]. Flink combines MapReduce functionality (e.g., schema flexibility and rich user defined functions) with techniques from traditional relational database management (e.g., query optimization, custom memory management, and pipelined processing) and adds dataflow and iterations. While having a different architecture, it offers similar functionality as Apache Spark and is, therefore, a candidate for a comparative benchmark implementation.
Quest is a massively distributed query processing engine offering from Metanautix, part of Microsoft. Quest is fully ISO/ANSI SQL'99 compliant, with a several extensions. It natively supports document data structures The Quest engine also connects to many data sources and extends the industry-standard Parquet columnar format with statistics for faster processing. User-defined functions can be written in LUA, C#, Java, Python, or SQL. A SQL extension, called Pipelines, is used to group SQL statements for more complex processing, such as the Pearson correlation, or K-Means (see Appendix A).

Prototype implementations of the benchmark on Flink and Quest, proves TPCx-BB is capable of working on non-Hadoop frameworks. TPCx-BB are open to new implementations, where TPCx-BB can be used to compare the performance and scalability of big data offerings and drive innovation in this space.

TPCx-BB in the cloud. At the high level TPCx-BB does not differentiate running the benchmark on SUT hosted in a datacenter or in the cloud. In the case of Infrastructure as a Service (IaaS) offerings from various cloud vendors, the benchmark can run with right framework and version requirements are met. In the past, the benchmark was run in Amazon AWS using different Hadoop distributions. However, on Big Data as a Service (BDaaS) offerings where the big data framework is an integrated offering, the benchmark is yet to be tested, examples of such offerings are Amazon Elastic MapReduce and the Databricks Cloud. For a fully valid result, where a test sponsors uses TPCx-BB on BDaaS for results publication, it should be noted, that the benchmark mandates adherence to the TPC pricing specification. TPC is working on amending their pricing specification to include cloud based offerings and facilitate cloud based TPC benchmark publications.

4 TPC Standardization of Big Bench

Founded in 1988, TPC's goal is to create, manage and maintain a set of fair and comprehensive benchmarks that enable end-users and vendors to objectively evaluate system performance under well-defined, consistent and comparable workloads. Currently, the TPC offers six are enterprise benchmarks (TPC-C and TPC-E for OLTP, TPC-DI for data integration, TPC-H for data warehouse, TPC-VMS for virtualization and TPC-DS for big data) and three are express benchmarks (TPCx-V for virtualization, TPCx-HS and TPCx-BB for big data). The TPC offers in parallel to the above listed benchmark specification so called Common Benchmarks, i.e. TPC-Energy and TPC-Pricing. These benchmark standards guarantee that energy consumption and pricing is measured in a consistent way across all performance benchmarks.
One of the pillars on which the credibility of TPC benchmarks rests is its strict audit rules. Audit rules guarantee that each benchmark publication was done according to its specification. TPCx-BB result is certified either by an independent certified TPC auditor or a TPCx-BB pre-publication board. The method to use is under the discretion of test sponsor.

4.1 Challenges during the Standardization

Standardizing an industry standard, involves framing set of rule and governance models. The process of standardization is a complex, cumbersome and time consuming process even for Greenfield benchmarks. Furthermore, the complexity was increased in the case of TPCx-BB where the specification had to consider the existing benchmark prototype during the process. This entire process posed unique set of challenges for the TPCx-BB sub-committee. The sub-committee worked diligently to address each of these issues, reached consensus and finally voted unanimously to launch benchmark.

In this section, we make an attempt to present few selected challenges occurred during the standardization process, addressing previously uncharted areas in any TPC specification.

Execution Rules. The benchmark specification defines a set of narrow rules to ensure the results are consistent with the standard, auditable by an independent auditor and close any potential for gaps, which could be exploited to create benchmark specials. In TPCx-BB run rules requires the benchmark to be run two times for performance and repeatability of the results. The lower (i.e., worse) result metric of the two runs is reported. Each run must include, Data generation, load test, power test, throughput test and result check. The benchmark also adds an additional test to validate the query answer set for consistency by running scale factor 1 on the SUT. The results along with supporting files are audited for correctness by a TPC auditor or the publication board before publishing the result. The sub-committee spent considerable time in providing various tuning, and optimization options for test sponsors to experiment and get the best results possible, without breaking any of the rules. In addition to tuning the framework, the benchmark kit provides run time tuning options at global level where the tuning parameters are applied for the benchmark as whole and tuning individual queries by passing explicit parameters for a query. The benchmark specification provides clearly defined areas with examples in the appendix for such tunings. In an effort to keep answer sets for consistent for engine validation test, the sub-committee has put in place a set of rules to accommodate the differences between various query engines. This helps not only addition of future frameworks, but also fast evolving SQL on Hadoop frameworks like Hive. The benchmark also applies TPC-Pricing specification where necessary, which is mandatory for published results and provides the option to report the TPC-Energy metric.

Scale Factor. TPCx-BB's data set scales linearly with the scaling factor (SF). In order to be realistic across a large bandwidth of data set sizes (1 GB to 1PB), the individual tables do scale in different ratios. While the large fact tables (sales and returns) scale linearly, other tables scale logarithmic or are completely static. Although this is realistic, it means that the ratio of sizes of the table changes with scale factors, e.g., for SF 1 the ratio of fact tables to dimension tables is approximately 50:50, while for large SFs the ratio becomes shifted to the fact tables. While BigBench scales continuously, TPCx-BB only specifies specific scale factors similar to TPC-H and TPC-DS (1, 3, 10, 30 …). Minor adjustments were made to the individual table scaling to ensure very close to linear scaling behavior for the full data set.

Metric. TPCx-BB's metric underwent a series of changes along with the execution model until its final version made it to the standard. The initially proposed metric was specified as the geometric mean of the execution time:

$$BB = \sqrt[4]{T_L * T_D * T_P * T_B} \qquad (1)$$

where T_L is the time taken for loading the data into the system, T_D is the time for declarative queries, T_P is the time taken to process all procedural queries, and T_B is the

time to process mixed queries. The query type is based on the implementation of the queries (declarative, procedural, or both).
However, since this is different for different kind of systems an alternative metric was proposed.

$$BB = \sqrt[30]{\prod_{i=1}^{30} P_i} \qquad (2)$$

which also uses the geometric mean, but rather than summing the queries according to the classes uses each processing time individually. Both metrics only consider a power test style setup, where each query is processed individually and do not account for multi stream setups, where multiple users submit queries to a system. Also, they measure the runtime directly, meaning a smaller result is better. To improve this, a new metric was proposed in [20], which changed from a geometric mean to an arithmetic mean for all parts and incorporated not only the stream use case (throughput test TT) but also a data maintenance step (DM). The metric is scaled by the number of streams (S) to compute the total number of queries processed per hour (3600 seconds) incorporating regular updates (individual times are measured in seconds):

$$BBQph = \frac{30*3*S*3600}{S*T_L + S*T_P + T_{TT_1} + S*T_D + T_{TT_2}} \qquad (3)$$

Although easy to understand, the arithmetic mean is not ideal in the case of highly skewed processing times. Since some queries process much less data than others and the data size processed does not scale linearly with the scaling factor for all queries, this is an issue in TPCx-BB. In this case, some queries will have very limited influence on the result of the metric. Therefore, a combination of geometric mean and arithmetic mean was finally incorporated in the standard:

$$\mathrm{BBQpm@SF} = \frac{\mathrm{SF} * 60 * \mathrm{M}}{\mathrm{T_{LD}} + \sqrt[2]{\mathrm{T_{PT}} * \mathrm{T_{TT}}}} \qquad (4)$$

The load time T_{LD} (reduced by a factor of 10) is added to the geometric mean of the power test time T_{PT} and the throughput test time T_{TT}. Again, all times are measured in seconds but the metric is reported per minute (60 seconds). The number of queries (M) is divided by the sum of load and processing time, in order to get larger results for larger scale factors, the metric is multiplied by the scale factor (SF). While the power test time is compute as the geometric mean of all individual query processing times, the throughput time is the total processing time of all streams divided by the number of streams. Although not as easy to understand as the second metric, the final metric finds a good compromise for enabling useful optimizations.

Machine learning techniques. Three queries in TPCx-BB implement clustering, regression, and classification at various stages to satisfy the use case requirements. The benchmark kit uses algorithms bundled with Apache MLlib to invoke machine learning stages. Differing from standard based SQL API's where answer sets can be matched with relative accuracy, in machine learning techniques it is expected to see changes in answer set for two reasons, a) changes to the algorithm in the same machine learning library for different versions, b) introduction of a new machine learning library which may use a different method to implement an algorithm. TPCx-BB being an end to end system performance benchmark, leaves validating accuracy of an algorithm outside the scope of the specification. However, foreseeing these issues the specification provides general guidelines to address answer set changes triggered by change in library versions. In case no other changes apart from library updates are in the code or parameters in the benchmark kit the results are consider as valid and the reference results can be updated. In the case of new machine learning library, the new implementation may modify the code and parameters in the benchmark kit, but needs to use the same input data set and needs to match or improve the algorithm accuracy provided in the existing library. TPCx-BB addresses these variations in the specification of machine learning for the first time and thus, eases extensions of the benchmark and integration of changes during the lifecycle of the benchmark.

Determinism Requirements. SQL queries written for benchmarks are typically reproducible. They always return the exact same result independent of the execution engine. This is an important requirement for auditing since it enables verifying the correctness of query results and ensures all SUTs actually have to perform the same work. TPCx-BB contains several non-SQL workloads, some of which are machine learning tasks. These are typically implemented in a non-deterministic way and different algorithms can produce different results. In fact, the result quality typically depends on the number of iterations an algorithm has run for (up to the maximum achievable quality for an algorithm). This is a challenge for performance benchmarking, since result quality can be traded for performance. To alleviate this problem the kits algorithms are designed in a way that they produce the exact same results, or – where this is not possible – other implementations' algorithm have to have at least the same quality as the default implementation.

Reaching consensus. Although BigBench was fully implemented in a kit when it was proposed to the TPC, the specification had to be extended to cover all required regulations and rules. In this process, multiple changes were introduced to, one the one hand, fix minor deficiencies and to, on the other hand, not penalize certain vendors that have slightly different / not completely compatible functionality. This is one of the most delicate parts of standardization, since disagreement on this level can delay or even stop a benchmark standardization. One of the more controversial topics during the standardization of BigBench was the metric, as briefly touch upon above. To solve this, the TPC subcommittee went through the process of preparing a model that can estimate performance, based on previously collected information, and using this to estimate the result of an execution. Being able to rethink and discuss setups with some numbers rather than on a theoretical level made it much easier for the committee to reach consensus.

5 Experiments with TPCx-BB Benchmark

In this section, we present experiments that were executed on independent test platforms, different frameworks, and small and large scale factors. We also discuss the

hardware resource utilization behavior of one of the test platforms. Table 1 shows test details of the experiments.
The test runs were conducted with default settings, except where parameters needed to be configured to ensure all queries are able to run successfully. The data set was generated using the default data generator and the tests were run using the driver provided in the kit.

Test #	Nodes in Cluster	Framework	Scale Factor
1	9	Hive on MapReduce	3000
2	8	Hive on Spark	1000
3	8	Hive on Tez	3000
4	8	SparkSQL	3000
5	1	Metanautix	1
6	8	Apache Flink	300
7	60	Hive on MapReduce	100000

Table 1 Test run experiments

5.1 Experimental Results

Test 1. The original implementation of the benchmark uses *Hive on MapReduce*. The test platform was configured with suitable parameters for Yarn, HDFS, and Hive, the benchmark was run with all three phases with two concurrent streams (default value) and completed successfully. Phase elapsed times were: load: 2803s, power: 34076s, and throughput: 54705s.

Test 2. *Hive on Spark* utilizes Apache Spark as execution engine for Hive. Hive on Spark reuses Hive's planner / optimizer. The primary benefit is that Hive on Spark automatically gets full compatibility with all of Hive's features. The benchmark can run with Hive on Spark, with small changes in the configuration and changes on the cluster to enable Hive to use Spark as the execution engine. All the three phases of the benchmark completed successfully on the test platform. Phase elapsed times are: load: 9389s, power: 13775s, and throughput: 13864s.

Test 3. *Tez* is designed to run batch and interactive workloads using the Hive API. In this test the load phase completed successfully, in the power phase 29 of 30 queries completed successfully. However Q16 failed to complete throwing an exception. The elapsed times for load was 3719s.

Test 4. *SparkSQL* is an offering from Apache Spark to process structured data. SparkSQL is compatible with Hive, making it possible to run queries written in HiveQL without modifications. Enabling SparkSQL support for all 30 queries has been a multi month effort, where the benchmark team worked with the Apache Spark community to identify and fix missing features and bugs that prevented the complete execution of TPCx-BB queries. In this test, we had to apply a patch to Spark version 1.6.1 to get all queries to run successfully. This patch should be made available in yet to release Spark version 2.0. All three phases of the benchmark completed successfully on the test platform. Phase elapsed times were: load: 7896s, power: 24,228s, and throughput: 40,352s.

Test 5. The *Metanautix* query processing engine is part of Microsoft's big data portfolio. All of the TPCx-BB queries were translated in SQL including sentiment analysis using a combination of window functions, user-defined Java functions, and pipelines. The machine learning post-processing stages were excluded.

Test 6. *Apache Flink* is a big data streaming dataflow processing engine compatible to the Hadoop stack. While having a different architecture with a purely stream-oriented execution engine, it offers similar functionality as Apache Spark. As a proof of concept, 22 queries were implemented using Flink's DataSet API. In order to cover all necessary machine learning capabilities, a Flink-backed SystemML implementation was used for two of the queries [11].

Test 7. The objective of this test to demonstrate readiness of the benchmark to scale beyond small dataset and clusters. For this purpose, we selected a cluster with 60 nodes and dataset scale factor of 100000 which is close to 100TB of input data. Hive on MapReduce was used as execution framework. We ran load and power phase and skipped the throughput phase due to limited availability of cluster time. Phase elapsed times were: load: 19,941s and power: 401,738s. During the tests, we found that the usage of realistic data distribution models in the benchmark result in a number of skewed tasks on Hive on MapReduce, where skewed tasks processes many more records than others and took much more time to complete. While this behavior is seen across all scale factors and cluster sizes, the result is amplified running the benchmark on the larger dataset and more number of nodes, challenging the efficiency of the query engine.
This set of experiments shows that various big data frameworks are able to run the benchmark with modification or no modifications, as demonstrated by experiments 1-6. This proves the versatility of the benchmark kit and shows that it can be used to compare and distinguish multiple frameworks for their features and performance. Partial execution of the kit on Metanautix shows that non-Hadoop-based frameworks are capable of adapting the benchmark. Partial execution of the benchmark on Apache Flink demonstrates the system agnostic nature of TPCx-BB, the use cases can be implemented natively without higher level SQL expression API's. Data and cluster scale tests bring out issues which are mostly uncaught during the development stages proving that a benchmark's role goes beyond providing publications but also helping vendors iteratively tune their platforms.

5.2 Resource Utilization Tests

Hardware platform tuning is often used to optimize the SUT to its maximum efficient state, i.e., the configuration where the test hardware is fully utilized with no obvious bottlenecks. Analysis of the hardware behavior under the load is crucial to understand the baseline performance and identify and resolve any bottlenecks. In this section, we analyze hardware resource utilization comparing the utilization patterns of the test platform by running the benchmark two times on a fixed hardware setup, scale factor, and big data framework. In the second test, we increase the number of concurrent streams in the throughput phase from 2 to 4.

Benchmark Setup. The cluster consists of eight HPE DL360 G8 nodes, with the configuration shown in Table 2. The experiments were conducted running all three phases of TPCx-BB on Scale Factor 3000. Hive on MapReduce was selected as the framework. Intel's Performance Analysis Tool[3] was used to collect the utilization pattern from the cluster nodes.

Node	Role	Hardware	Software
1	Master Server	24C,192GB RAM, 8.5TB storage, 10Gbe	RHEL 6.7, CDH 5.6
2-8	Worker Node	24C,256GB RAM, 8.5TB storage, 10Gbe	RHEL 6.7, CDH 5.6

Table 2 Cluster configuration

Table 3 shows the elapsed times for load, power, and throughput phase for both of the test runs. The load phase consists of reading the generated data to create the test dataset in appropriate format; copy data into final location; data preparation including metadata creation, population, and computation of database statistics. The power phase is designed to measure the performance of the SUT when processing all the queries in sequential order. The elapsed times for load and power phases are comparable with variation expected from a Hadoop system. In this test we are in particular interested in the system characteristics of the throughput phase. During this phase, all queries are executed using concurrent streams. Each query stream runs all queries, where each stream has a different order of queries. As can be seen in the table, the elapsed time for the throughput phase doubles for 4 concurrent streams in comparison to 2 concurrent streams.

Phase	2 Streams	4 Streams
Load	2803	2796
Power	34076	34179
Throughput	54705	104565

Table 3 Elapsed times

Analysis of Utilization Pattern. The charts in Figures 2 show the hardware utilization pattern behavior of the cluster when running the benchmark with 2 and 4 streams. The chart shows comparison of the major components of the cluster, i.e., CPU utilization, memory utilization, I/O bandwidth, and network I/O. Since we have captured data at one second samples, the chart is compressed on the time scale to show the complete execution of the benchmark.

[3] PAT - https://github.com/intel-hadoop/PAT

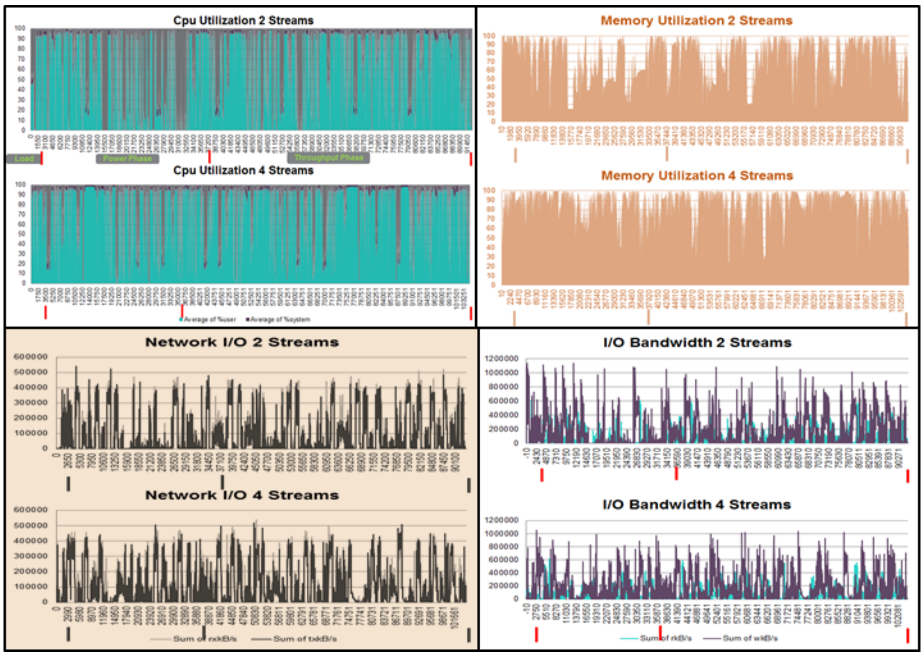

Figure 2 Processor, memory and I/O utilization

The first mark in the time scale in Figure 2 marks the end of the load phase of both test which is at ~ 2,800s. The load phase involves data staging and replicating over data nodes that results in a cluster management overhead. This governs the performance of this stage with significant CPU and memory utilization, I/O bandwidth, and network I/O. The load phase uses software compression to compress the raw input data into optimized columnar format, resulting in additional CPU utilization.

The power phase utilization can be seen between the first and second mark in the time scale in Figure 2. The individual peaks are signatures of each query being run in sequential order. Additional insight of the queries can be gained by mapping the running time of each to the time dimension on the charts. The independent utilization pattern for each query highlights that, unlike the constant ramp-up and down seen in micro-benchmarks, TPCx-BB exhibits use case driven utilization patterns close to real world big data use cases, where the platform needs to accommodate both short and long running tasks. It is possible to go into more fine granular analysis of each query and gain insight into the individual system resource usage. This leads to a better understanding of the query and system behavior when tuning individual queries. As expected the power phase shows very similar comparative elapsed times between the two tests.
In Figure 2, the second mark indicates the start of the throughput phase, the CPU utilization shows a steady high processor usage. A more detailed analysis showed 70 % utilization for two concurrent streams and 90% utilization for four concurrent streams. The memory, storage, and network I/O are sufficiently utilized but nowhere close to the processor utilization. We can estimate the overhead effect when observing the ratio of the throughput phase execution time. As the number of streams doubled from 2 to 4,

the execution time increases by a factor of 2. The overhead of running more streams can be inferred by varying the number of streams. The throughput phase reflects the nature of big data workloads comprising a mix of both short running and long running tasks executing side by side on a cluster [12].
The emphasis of TPCx-BB to simulate real-world scenario for big data batch analytics helps to extrapolate the findings and apply the takeaways when deploying big data applications. By running the above experiments we summarize few key takeaways:

- When selecting the hardware for big data clusters, it is important to evaluate computing power, memory capacity, storage, and network bandwidth in conjunction with intended data set size and number of tasks required to run side by side.
- Contrary to common belief that big data workloads are I/O bound, we notice – with an adequate I/O setup – big data workloads tend to be compute bound. Similar results are also reported by [13,21] during their independent tests.
- Efficient utilization of hardware resources highly depends on framework tuning. In this example, we believe – as software schedulers evolve – the utilization pattern of peaks and valleys of will reduce when freeing hardware resources and reducing the wait times for waiting queued tasks.
- Selective utilization of accelerators and off-load engines could be beneficial to increase overall efficiency of the cluster. An example could be load phase compression off-load.

6 Benchmarking Emerging Big Data Use Cases

In recent years, there have been large advances in analytics software. As big data reaches a larger audience, the community has sought to commoditize general purpose algorithms and systems for increasingly elaborate analytical tasks. The generation of large datasets has been increasing, leading to the development of new big data processing frameworks, which is predominantly driven by “People and Things”. For example, “People” interacting via social media portals and cloud enabled applications are driving an ever increasing volume of data into the cloud [14]. “Things” are intelligent and connected devices capable of making semi-autonomous decisions using models received by cloud-based or -hosted compute farms. Addressing these two important segments with a relevant benchmark, will help the industry and academic community to validate the performance of new implementations.
There is a broad range of new applications for these analytical capabilities, to name a few:

- Recommendation systems: graph processing, stream processing, machine learning.
- Search and ranking: graph processing, machine learning
- Fraud detection: machine learning, *ad-hoc* analysis.
- Internet-of-Things (IoT): stream processing, lambda processing.
- Image, video, audio, and natural language processing: deep learning using neural networks

For the purpose of this paper and benchmarking, we select two categories of the advances as follows:

- Processing frameworks
- Machine learning

6.1 Processing frameworks

Stream. Stream processing is mainly used in real-time analytics, where the events are streamed in form of micro or mini batches. A data stream can be as simple as time series events displayed in real-time, e.g., temperature readings from a sensor, or processed as complex events by applying computation techniques in real-time, e.g., identifying failed components in an airplane using anomaly detection techniques. In addition to acting on the incoming stream in real-time, events are stored for feedback-based learning and historical trend analysis using batch analytics.

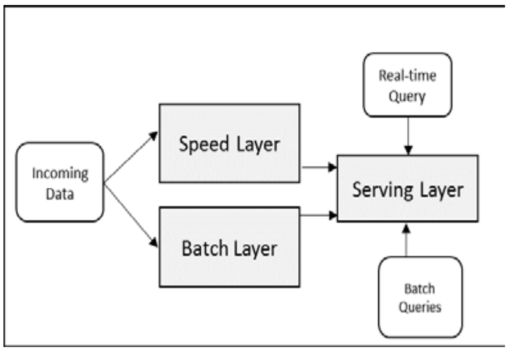

Figure 3 Lambda Architecture

Data from a device in the field can be permuted and aggregated at the source or in midway before it is transferred to the cloud. The lambda architecture [15] is an example of a stream processing framework using three layers of processing.

- Batch Layer - curates the master dataset by storing all data entering the system using batch processing techniques.
- Serving Layer – enables fast *ad-hoc* insights extracted from data curated in the batch layer.
- Speed Layer – provides real-time insights from the incoming/streamed data, including running machine learning algorithms, on real-time data.

An IoT benchmark based on such an architecture can serve as an excellent proxy to test functions involved with streaming and real-time analytics.

Graph. Human interaction with the internet changed the Web 2.0 [16]. The emergence of various social networking platforms, search engine optimizations, and the ability to connect these human interactions with business models was unthinkable just a decade ago. Graph processing systems are used in analyzing networks of relationships normally represented in

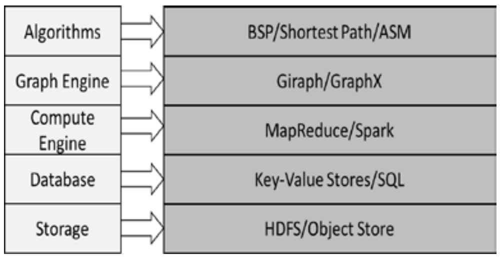

Figure 4 Graph processing framework

data objects referred as nodes and edges. Some large graph datasets can span trillions of edges [17].
Graph processing requires a robust framework with characteristics such as, fault tolerant storage, fast database, scale-out graph analysis engines, scale-out computation engine, and efficient algorithms as illustrated in Figure 3.

6.2 Machine learning

Machine learning techniques continue to grow in significance but also are expanding into different areas of application. With this growth the field is transitioning from a few "bespoke" applications; e.g., image recognition, machine translation, speech recognition, and robotics, to more commoditized ones; e.g., fraud detection. We will focus on the latter, which typically operate on discrete symbols such as words as opposed to continuous input such as from a microphone or historical revenue.
Machine learning has a broad range of applications with different algorithms being employed. These algorithms typically fall into two categories:

1. Regression, which works to predict a variable's value (e.g., projection of revenue),
2. Classification, is concerned with predicting a label for a sample (e.g. male/female, will or will not buy).

Moreover, a task can be structured where the prediction happens on a graph or sequence such as machine translation generating a sequence of words in a foreign language. The task can also be unstructured, where the desired output is a single value like the next stock price, or whether a fraud occurred or not. Training of a model can be supervised, unsupervised, or utilizing reinforcement learning. In each of these scenarios, one can define a measure of quality such as in the case of fraud detection;

1. A weighted sum of false positives - fraud was declared when a transaction was in good standing
2. False negatives - fraud remained undetected

Because the data are generated automatically, they have special properties which can be exploited by the algorithms. Therefore, as in TPCx-BB, we should factor in the speed of the algorithms.
TPCx-BB as a batch analytics benchmark provides excellent coverage for advanced analytics to examine large datasets. Most of the benchmark is implemented using data management primitives and functions. Although there are a handful of use cases in TPCx-BB invoking machine learning algorithms[4], TPCx-BB is far from being a comprehensive representation of analytics using machine learning algorithms. Currently, neither streaming processing, graph processing, nor deep learning are represented in TPCx-BB. Given the recent interest in deep learning, and its broad range of applicability, it should be given special consideration.
There have been some efforts in the analytics community to address these areas [18, 19]. However, there hasn't been any collaborative push from the industry and academia to create a use case based benchmarking framework. We think it would be impractical

[4] Examples are clustering, logistic regression, and sentiment analysis.

to expand the coverage of the TPCx-BB benchmark to include all of these, therefore, they should be the focus of future benchmarks.

7 Conclusion

In 2013, the proposal “BigBench” was brought to the attention of the analytics community as a candidate for a first end-to-end big data benchmark. Since then idea has evolved, been put under the scrutiny of experts and public alike to finally emerge as TPCx-BB, the first industry standard big data benchmark with relevance to big data use cases. During this process, several changes went into the benchmark, which we discussed in this paper. Preliminary results are encouraging and it already has seen adoption with first results being published[5]. The benchmark helps the big data software ecosystem to identify performance bottlenecks, feature gaps, and scaling issues, which previously often remained undiscovered. The benchmark has also helped driving innovation in non-Hadoop ecosystems.
In this paper, we have tracked the course of the BigBench journey, gave a snapshot of its current state and potential changes coming in the future. We have conducted extensive experiments using the benchmark, and offered observations and analyses of several platforms. This paper offers a glimpse of the TPC standardization process, challenges and means to navigate through them successfully.

Acknowledgements. We would like to thank Sreenivas Viswanada from Microsoft Corporation for running experiments on Metanautix. Yao Yi and Zhou Yi from Intel Corporation for their help to run 100TB experiment. Michael Frank and Manuel Dansich from bankmark for their work on the TPCx-BB benchmark kit.
This work has been partially supported through grants by the German Ministry for Education and Research as Berlin Big Data Center BBDC (funding mark 01IS14013A) as well as through grants by the European Union’s Horizon 2020 research and innovation program under grant agreement 688191.

8 References

1. Frank McSherry, Michael Isard, and Derek G. Murray: Scalability! But at what COST? In HotOS ’15.
2. Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. BigBench: towards an industry standard benchmark for big data analytics. In SIGMOD ’13.
3. Raghunath Othayoth Nambiar, Meikel Poess, Akon Dey, Paul Cao, Tariq Magdon-Ismail, Da Qi Ren, Andrew Bond: Introducing TPCx-HS: The First Industry Standard for Benchmarking Big Data Systems. TPCTC ’14.
4. Meikel Poess, Raghunath Othayoth Nambiar, and David Walrath. Why you should run TPC-DS: a workload analysis. VLDB ’07.

[5] Hewlett Packard Enterprise ProLiant DL for Big Data – http://www.tpc.org/3501

5. Tilmann Rabl, Chaitanya Baru, Milind Bhandarkar, Meikel Poess, and Raghunath Nambiar. Setting the Direction for Big Data Benchmark Standards. In TPCTC ’12.
6. Devadutta Ghat, David Rorke, and Dileep Kumar. New SQL Benchmarks: Apache Impala (incubating) Uniquely Delivers Analytic Database Performance. [online]. https://blog.cloudera.com/blog/2016/02/new-sql-benchmarks-apache-impala-incubating-2-3-uniquely-delivers-analytic-database-performance/
7. Transaction Processing Performance Council. TPC Express Benchmark™ BB. [online] http://www.tpc.org/tpcx-bb
8. Chaitan Baru, Milind Bhandarkar, Carlo Curino, Manuel Danisch, Michael Frank, Bhaskar Gowda, Jie Huang, Hans-Arno Jacobsen, Dileep Kumar, Raghunath Nambiar, Meikel Poess, Francois Raab, Tilmann Rabl, Nishkam Ravi, Kai Sachs, Lan Yi, and Choonhan Youn An Analysis of the BigBench Workload. In TPCTC ’14.
9. Tilmann Rabl, Michael Frank, Hatem Mousselly Sergieh, and Harald Kosch. A data generator for cloud-scale benchmarking. In TPCTC ’10.
10. Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix Naumann, Mathias Peters, Astrid Rheinländer, Matthias J Sax, Sebastian Schelter, Mareike Höger, Kostas Tzoumas, Daniel Warneke: The Stratosphere Platform for Big Data Analytics. VLDB Journal 2014. Volume 23(6), pages 939-964.
11. Matthias Boehm, Douglas Burdick, Alexandre V. Evfimievski, Berthold Reinwald, Prithviraj Sen, Shirish Tatikonda, and Yuanyuan Tian. Compiling machine learning algorithms with SystemML. In SoCC ‘13.
12. Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy Katz. The Case for Evaluating MapReduce Performance Using Workload Suites. In MASCOTS '11.
13. Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung-Gon Chun. Making Sense of Performance in Data Analytics Frameworks. In NSDI ’15.
14. Daniel E. O'Leary. ‘Big Data’, the ‘Internet of Things’ and the ‘Internet of Signs’. In Intelligent Systems in Accounting, Finance and Management. Volume 20(1), pages 53-65.
15. Nathan Marz and James Warren. Big Data: Principles and best practices of scalable realtime data systems. Manning Publications. 2015.
16. Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. SIGMOD ’10.
17. Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One Trillion Edges: Graph Processing at Facebook-Scale. PVLDB 8(12): 1804-1815 (2015).
18. Min Li, Jian Tan, Yandong Wang, Li Zhang, and Valentina Salapura. SparkBench: a comprehensive benchmarking suite for in memory data analytic platform Spark. In CF ’15.
19. Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. In SoCC ’10.
20. Tilmann Rabl, Michael Frank, Manuel Danisch, Bhaskar Gowda, and Hans-Arno Jacobsen. Towards a Complete BigBench Implementation. In WBDB ’14.
21. Yanpei Chen, Alan Choi, Dileep Kumar, David Rorke, Silvius Rus, and Devadutta Ghat. How Impala Scales for Business Intelligence: New Test Results. [online]. http://blog.cloudera.com/blog/2015/09/how-impala-scales-for-business-intelligence-new-test-results/

9 Appendix A

K-Means using SQL. It is possible to write K-means using SQL and extensions in the Metanautix Quest system. The full implementation is complex, requiring an iteration (implemented using SQL triggers), but also rebalancing when a class becomes empty. For simplicity we assume that each point is described by an id, and a coordinate vector x. Using a SQL UDF, we can write the Distance function. A user-defined aggregation function, AVG_VECTOR, computes the average vector. We assume 50 classes. We outline the steps:

1. Initialization of class centroids

```
CREATE TABLE Centroids .. AS
  SELECT ROW_NUMBER() OVER (ORDER BY RANDOM()) r, x FROM
Data WHERE r <= 50
```

2. Assigning data points to classes

```
CREATE TABLE ClassAssignment .. AS
 SELECT id, r FROM Centroids C, Data D WHERE
    Distance(D.x, C.x) = (SELECT MIN(Distance(D.x, C2.x))
FROM Centroids C2)
```

3. Compute new centroids

```
CREATE TABLE NewCentroids .. AS
  SELECT r, AVG_VECTOR(x) x FROM Centroids C, ClassAs-
signment CA, Data D WHERE
    C.r = CA.r AND CA.id = D.id
```

Using window functions. Window functions can be used where a MapReduce, or multiple passes would be otherwise required. As an example, we show how Query 02 can be rewritten.

```
WITH Session as (
SELECT DISTINCT
  sessionid,
  wcs_item_sk
FROM
(SELECT
  *,
  concat(cast(wcs_user_sk as string), '_', cast(bucket as
string)) sessionid
FROM
(SELECT
  *,
  (first(tstamp_inSec) over (partition by wcs_user_sk
                              order by tstamp_inSec desc)
- tstamp_inSec) / 3600 bucket
FROM
  (SELECT
    wcs_user_sk,
    wcs_item_sk,
    (wcs_click_date_sk * 24 * 60 * 60 +
wcs_click_time_sk) AS tstamp_inSec
  FROM web_clickstreams
  WHERE wcs_item_sk IS NOT NULL
  AND   wcs_user_sk IS NOT NULL))))
```