Towards A Complete BigBench Implementation

Tilmann Rabl[1,2], Michael Frank[2], Manuel Danisch[2], Bhaskar Gowda[3], and Hans-Arno Jacobsen[1]

[1] Middleware Systems Research Group
[2] bankmark UG
[3] Intel Corporation
tilmann.rabl@utoronto.ca, michael.frank@bankmark.de, manuel.danisch@bankmark.de, bhaskar.d.gowda@intel.com, jacobsen@eecg.toronto.edu

Abstract. BigBench was the first proposal for an end-to-end big data analytics benchmark. It features a set of 30 realistic queries based on real big data use cases. It was fully specified and completely implemented on the Hadoop stack. In this paper, we present updates on our development of a complete implementation on the Hadoop ecosystem. We will focus on the changes that we have made to data set, scaling, refresh process, and metric.

1 Introduction

Modern storage technology enables storing more and more detailed information. Retailers are able to record all user interaction to improve shopping experience and marketing. This is enabled by new big data management systems. These are new storage systems that make it possible to manage and process ever larger amounts of data. To date a plethora of different systems is available featuring new technologies and query languages. This makes it hard for customers to compare features and performance of different systems. To this end, BigBench, the first proposal for an end-to-end big data analytics benchmark [1] was developed. BigBench was designed to cover essential functional and business aspects of big data use cases.

In this paper, we present updates on our alternative implementation of the BigBench workload for the Hadoop eco-system. We reimplemented all 30 queries and the data generator. We adapted metric, scaling, data set, and refresh process the fit the needs of a big data benchmark.

The rest of the paper is organized as follows. In Section 2, we present a brief overview of the BigBench benchmark. Section 3 introduces the new scaling model for BigBench. We give details on refresh process in Section 4. Section 5 presents our proposal for a new big data metric. Section 6 gives an overview of related work. We conclude with future work in Section 7.

2 BigBench Overview

BigBench is based on the Transaction Processing Performance Council's (TPC) new decision support benchmark TPC-DS [2, 3]. TPC-DS models a retail business with store,

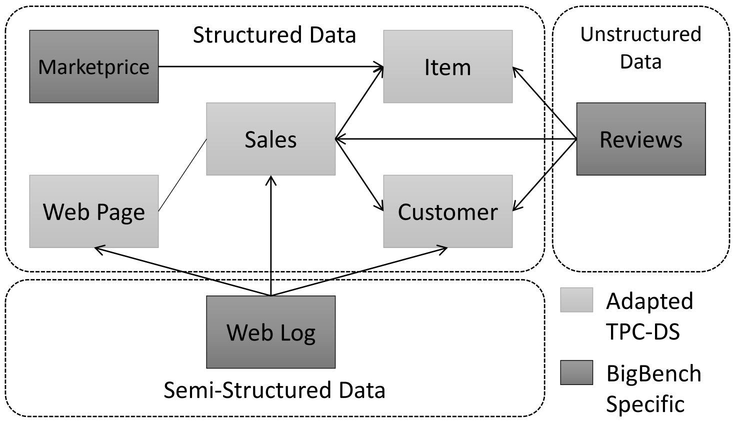

Fig. 1. BigBench Schema

online, and catalog market channels. For BigBench the catalog channel was dropped and additional big data related information sources were added. The reason for this was that catalog sales are becoming ever less important in current retail businesses and more and more additional information sources are included in big data analytics. A high-level overview of the extended data model can be seen in Figure 1. The picture shows the sales channel, which in the full schema consists of web sales, web returns, store sales, and store returns. The dimension marketprice was added to store competitors' prices. The Web Log portion represents a click stream that records user behavior on the online presence. The click stream is stored in comma separated value form but resembles a web server log. The product reviews contain full text that is used for natural language processing. The complete schema is described in [4]. In this new version of the data generator, the complete data set was implemented using the parallel data generation framework [5].

BigBench features 30 complex queries, 10 of which are taken from TPC-DS. The queries are covering the major areas of big data analytics [6]. As a result, they cannot be expressed by pure SQL queries. The full list of queries in Teradata Aster SQL-MR syntax can be found in [4]. In the current version, all queries were implemented using Hadoop, Hive, Mahout, OpenNLP [7].

3 BigBench Scaling

As the original BigBench implementation was partially based on the TPC-DS data generator DSDGen, it was bound to the same restrictions in terms of scaling. DS achieves perfect data size scalability, i.e., data is exactly scaling as specified by the scale factor scale. This is done by restricting the scale factor to certain numbers (100, 300, 1000, 3000, 10000, 30000, 100000) and fine tuning the table sizes to the correct data size for each scale factor. This limits the maximum data size to 100TB. Furthermore, there are no intermediate data sizes, which makes it hard to test load boundaries of a system or testing the optimal load factor. Therefore, we have redesigned the scaling to allow for a continuous scaling.

Simply using a linear scale factor for all table sizes is not an option for BigBench, since it is supposed to scale across a very wide range of scale factors. Our initial target is to have reasonable table sizes for all tables for the range of 1 GB to 1 PB, allowing for small and fast experimental tests up to large scale scale out experiments. This means that the number of entries of a linearly scaled table will be increased by a factor of 10^6 for the 1 PB data set. This is not realistic for many of the tables, for example, it is unlikely that a company has more than 50K stores worldwide. For comparison, Walmart, the world's largest retailer has 11K stores[4], McDonald's has 33K restaurants worldwide[5], Starbucks has 17K stores worldwide[6]. Consequently, several tables in the schema cannot be scaled linearly. TPC-DS uses linear, logarithmic, and square root order of growth for different tables but the individual numbers are hand adjusted. We use the same approach but uniformly adjust the linear scaling factors to make up for the non-linear growth of other tables. An overview of all tables and their growth factors can be seen in Table 1.

Table Name	# Rows SF 1	Bytes/Row	Scaling
date	109573	141	static
time	86400	75	static
ship_mode	20	60	static
household_demographics	7200	22	static
customer_demographics	1920800	40	static
customer	100000	138	square root
customer_address	50000	107	square root
store	12	261	square root
warehouse	5	107	logarithmic
promotion	300	132	logarithmic
web_page	60	134	logarithmic
item	18000	308	square root
item_marketprice	90000	43	square root
inventory	23490000	19	square root * logarithmic
store_sales	810000	143	linear
store_returns	40500	125	linear
web_sales	810000	207	linear
web_returns	40500	154	linear
web_clickstreams	6930000	27	linear
product_reviews	98100	670	linear

Table 1. Overview of Table Growth Factors

[4] Walmart Interactive Map - `http://corporate.walmart.com/our-story/our-business/locations/#/`

[5] McDonald's FAQs `http://www.mcdonalds.ca/ca/en/contact_us/faq.html`

[6] Starbucks Company Profile - `http://www.starbucks.com/about-us/company-information`

To adjust the sublinear growth of some of the tables, we increase the linear scaling factor by the percentage that is missing to get a linear scaling of the data size. This can be computed by measuring the relative data size of each different class of growth factor for scale factor 1. For example, the linearly scaled tables are contributing roughly 50% of the data to the 1 GB data set. The rest comes from static and sublinearly scaling tables. The strongest growing table of the latter is inventory, which contains a record for every item in every warehouse every day. To make up for the missing data size we compute the difference in growth between the linear factor and the sublinear factors in comparison to the base size and increase the linear factor by this factor:

$$LF = SF + (SF - (\log_5(SF) * \sqrt{SF})) = 2SF - \log_5(SF) * \sqrt{SF} \qquad (1)$$

Where LF is the linear factor and SF is the scale factor. For large scale factors the linear factor converges towards $2SF$ and thus the linearly scaling tables make up almost the complete data size as expected.

4 Refresh

The refresh process initially specified in BigBench is an exact copy of the TPC-DS refresh [1]. TPC-DS mandates the refresh to happen during a throughput test. Given S streams of query sequences, each simulating a user, there are $S/2$ refresh operations each scheduled after 198 queries are completed in total in the streams (each stream runs 99 queries for a total of $99 * S$ queries in all streams). This was added to TPC-DS later to ensure that systems are able to deal with trickle updates and do not "over-optimize" the storage layout. Many big data systems, hovever, process data in a batch oriented fashion. In this model, data is either completely loaded fresh after new data has to be processed or updates are processed in bulk as is also common in many data warehouses. In current Hive, for instance, any update of the data means overwriting complete files, since files in HDFS are immutable. Therefore, we have changed the refresh model to a single bulk update in between two throughput runs. This ensures that refresh has to be handled by the system (even if it means a complete reload of the data) but at the same time the refresh will not enforce stopping a throughput run, which would be the case using TPC-DS' update mechanism. By default, new data in the size of 1% of the original data set is inserted in the refresh phase. The size of the refresh data for each table is determined by the scaling model described in Section 3.

5 Metric

We propose a new metric for BigBench to take the batch-oriented processing of many big data systems into account and to include all parts of the benchmark. As BigBench aims at becoming an industry standard benchmark, we require a combined metric that returns a single (abstract) value that can be used to compare the entire end-to-end performance of big data analytics systems. The initially proposed metric for BigBench was loosely based on the TPC-DS metric. It consisted of four measurements:

T_L: Execution time of the loading process;
T_D: Execution time of the declarative queries;
T_P: Execution time of the procedural queries;
T_M: Execution time of queries with procedural and declarative aspects.

The complete metric was specified as the geometric mean of these four measurements:

$$\text{Metric} = \sqrt[4]{T_L * T_D * T_P * T_M} \tag{2}$$

The intuition behind the metric was that some systems are optimized for declarative queries while others are optimized for procedural queries and the classification can give an indication on the type of system that is benchmarked. There are a couple of problems with this metric, most notably the classification of queries, which is debatable. Because the concrete implementation of queries is not enforced the classification does not uniformly apply. For example, in Hive all queries will be transformed to MapReduce jobs and thus finally be run in a procedural way. Furthermore, the metric does not consider the refresh process or parallel execution of queries. To address these issues, we have revisited the TPC-H [8] and TPC-DS metrics [3]. To explain the reasoning behind the new metric, we will give a short overview of these two metrics as specified in the current versions. Both, TPC-H and TPC-DS specify two performance tests, a power test and a throughput test. The resulting numbers are meant to be comparable for a certain database size. The power test represents a single user run and shows the ability of a system to run a single stream of queries. The throughput test represents a multi user scenario where many queries are executed in parallel.

In TPC-H the power test is specified as a serial run of all 22 queries and one refresh process before and one after the query run. The power metric is defined as follows:

$$\text{Power\_Metric@Size} = \frac{3600 * SF}{\sqrt[24]{T_{R_1} * \prod_{i=1}^{22} T_{Q_i} * T_{R_2}}} \tag{3}$$

Where T_x is the processing time of a query or refresh function in seconds. The total metric is the geometric mean of the number of queries and refreshes that can be processed per hour multiplied by the scale factor. The reasoning behind multiplying with the scale factor is probably a higher comparability between different scale factors, since the data size in GB equals the scale factor. The throughput test is specified as a run of S serial streams of each a permutation of the 22 queries and a separate stream of $2S$ refresh functions. The scheduling of the refresh stream is not specified. The throughput metric is specified as follows:

$$\text{Throughput\_Metric@Size} = \frac{S * 22 * 3600 * SF}{T_S} \tag{4}$$

Where S is the number of streams and T_S is the total execution time of the throughput test. The metric is similar to the power metric but it computes the arithmetic mean of the processing times. It is unclear why the refresh functions are not considered in the arithmetic mean. Again the resulting average number of queries per hour is multiplied with the scale factor.

TPC-H does not consider the loading time as part of the metric. The final metric is specified as the geometric mean of the power metric and the throughput metric:

$$\text{Queries\_per\_hour@Size} = \sqrt{\text{Power\_Metric@Size} * \text{Throughput\_Metric@Size}} \tag{5}$$

TPC-DS has the same possible variables, the number of streams in the throughput test S and the scale factor SF. The TPC-DS metric consists of 3 parts:

T_{LD}: The load factor;
T_{PT}: The power test factor;
T_{TT}: The throughput factor.

The load factor is computed by measuring the time it takes to load the data T_{load} multiplied with the number of stream S in the throughput test and a factor 0.01: $T_{LD} = T_{\text{load}} * S * 0.01$.

The power test is a serial run of all 99 queries of TPC-DS. There is no data refresh in the power test in TPC-DS. The power test factor is the total execution time of the power test multiplied by the number of streams: $T_{PT} = T_{\text{Power}} * S$.

The throughput test consists of two independent, consecutive runs of S parallel query streams interspersed with $S/2$ refresh functions. Each of the streams executes all 99 queries in a permutation specified by TPC-DS. The reported time is the total execution time of both runs. All times are reported in hours.

The complete DS metric is specified as follows:

$$\text{QphDS@SF} = \left\lfloor \frac{SF * 3 * S * 99}{T_{LD} + T_{PT} + T_{TT}} \right\rfloor \tag{6}$$

Note that this metric is different than the initially presented metric for TPC-DS as specified in [2]. The initially specified metric did not contain the power test and specified a separate data maintenance phase between two query runs:

$$\text{QphDS@SF}_{\text{old}} = \left\lfloor \frac{SF * S * 198}{T_{LD} + T_{TT_1} + T_{DM} + T_{TT_2}} \right\rfloor \tag{7}$$

Where T_{TT_1} and T_{TT_2} are the run times of the 2 throughput test runs and T_{DM} is the run time of the data maintenance. This metric was designed to compute the average query throughput per hour including maintenance tasks. Based on these two metrics, we have designed a new metric that takes the application scenario of BigBench into account. The metric includes the following components:

T_L: Execution time of the loading process;
T_P: Execution time of the power test;
T_{TT_1}: Execution time of the first throughput test;
T_{DM}: Execution time of the data maintenance task.
T_{TT_2}: Execution time of the second throughput test;

All times are measured in seconds in at least 0.1 precision. In TPC-H loading is not measured at all, which is only valid if loading is a very rare activity in the life-cycle of a database. In TPC-DS a 1% fraction of the load time is incorporated in the metric, which is then multiplied by the number of streams to increase the impact of the effort of loading for increasing number of streams. This still implies loading is a rare operation in the targeted application. Since we believe that loading is an essential part of the workload in big data applications, we keep the full loading time in the metric. The finally reported metric is similar to the initial metric of TPC-DS:

$$\text{BBQpH} = \frac{30 * 3 * S * 3600}{S * T_L + S * T_P + T_{TT_1} + S * T_{DM} + T_{TT_2}} \tag{8}$$

The metric reports the mean queries per hour including the loading time and data maintenance time. The times are multiplied with the number of streams in order to keep the impact of the power test, the loading, and the data maintenance stable. The total number of queries run is $30 * (2 * S + 1)$, but since the run time of the power test is multiplied by S, the 30 queries in that run are counted S times. Therefore the numerator of the metric is $30 * 3 * S$ times the number of seconds in an hour.

6 Related Work

Today, TPC benchmarks are commonly used for benchmarking big data systems. For big data analytics, TPC-H and TPC-DS are obvious choices and TPC-H has been implemented in Hadoop, Pig[7], and Hive[8] [9, 10]. Subsets of TPC-DS queries have been implemented in Impala[9], Hive, Hawq, Shark, and others. TPC-H and TPC-DS are pure SQL benchmarks and do not cover all the different aspects that MapReduce systems are typically used for.

Several proposals try to modify TPC-DS similar to BigBench to cover typical big data use cases. Zhao et al. propose Big DS, which extends the TPC-DS model with social marketing and advertisement [11]. To resemble the ETL part of a big data workload, Yi and Dai have modified TPC-DS as part of the HiBench suite [12, 13]. The authors use the TPC-DS model to generate web logs similar to BigBench and use custom update functions to simulate an ETL process. Like BigDS this is orthogonal to BigBench and can be included in future versions of the benchmark. There have been several other proposals like the Berkeley Big Data Benchmark[10] and the benchmark presented by Pavlo et al. [14]. Another example is BigDataBench, which is a suite similar to HiBench and mainly targeted at hardware benchmarking [15]. Although interesting and useful, both benchmarks do not reflect an end-to-end scenario and thus have another focus than BigBench.

[7] https://issues.apache.org/jira/browse/PIG-2397
[8] https://issues.apache.org/jira/browse/HIVE-600
[9] http://blog.cloudera.com/blog/2014/01/impala-performance-dbms-class-speed/
[10] https://amplab.cs.berkeley.edu/benchmark/

7 Conclusion

BigBench is the only fully specified end-to-end benchmark for big data analytics currently available. In this paper, we present updates on the data model, refresh, metric, and scaling. The queries and the data set can be downloaded from GitHub[11].

References

1. 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)
2. Pöss, M., Nambiar, R.O., Walrath, D.: Why You Should Run TPC-DS: A Workload Analysis. In: VLDB. (2007) 1138–1149
3. Transaction Processing Performance Council: TPC Benchmark H - Standard Specification (2012) Version 2.17.0.
4. Rabl, T., Ghazal, A., Hu, M., Crolotte, A., Raab, F., Poess, M., Jacobsen, H.A.: BigBench Specification V0.1. In: Specifying Big Data Benchmarks. (2014) 164–201
5. Rabl, T., Frank, M., Sergieh, H.M., Kosch, H.: A Data Generator for Cloud-Scale Benchmarking. In: TPCTC '10. (2010) 41–56
6. 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`.
7. Chowdhury, B., Rabl, T., Saadatpanah, P., Du, J., Jacobsen, H.A.: A BigBench Implementation in the Hadoop Ecosystem. In: Advancing Big Data Benchmarks - Proceedings of the 2013 Workshop Series on Big Data Benchmarking. (2014)
8. Transaction Processing Performance Council: TPC Benchmark DS - Standard Specification (2013) Version 1.1.0.
9. Moussa, R.: TPC-H Benchmark Analytics Scenarios and Performances on Hadoop Data Clouds. In Benlamri, R., ed.: Networked Digital Technologies. Volume 293 of Communications in Computer and Information Science. Springer Berlin Heidelberg (2012) 220–234
10. Kim, K., Jeon, K., Han, H., gyu Kim, S., Jung, H., Yeom, H.: MRBench: A Benchmark for MapReduce Framework. In: Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on. (Dec 2008) 11–18
11. Zhao, J.M., Wang, W., Liu, X.: Big Data Benchmark - Big DS. In: Proceedings of the Third and Fourth Workshop on Big Data Benchmarking 2013. (2014) (in print).
12. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis. In: ICDEW. (2010)
13. Yi, L., Dai, J.: Experience from Hadoop Benchmarking with HiBench: from Micro-Benchmarks toward End-to-End Pipelines. In: Proceedings of the Third and Fourth Workshop on Big Data Benchmarking 2013. (2014) (in print).
14. 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 '09: Proceedings of the 35th SIGMOD international conference on Management of data. (2009) 165–178
15. 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: Proceedings of the 20th IEEE International Symposium On High Performance Computer Architecture. HPCA (2014)

[11] `https://github.com/intel-hadoop/Big-Bench`