Big Data Generation

Tilmann Rabl and Hans-Arno Jacobsen

Middleware Systems Research Group
University of Toronto
tilmann.rabl@utoronto.ca, jacobsen@eecg.toronto.edu
http://msrg.org

Abstract. Big data challenges are end-to-end problems. When handling big data it usually has to be preprocessed, moved, loaded, processed, and stored many times. This has led to the creation of big data pipelines. Current benchmarks related to big data only focus on isolated aspects of this pipeline, usually the processing, storage and loading aspects. To this date, there has not been any benchmark presented covering the end-to-end aspect for big data systems.
In this paper, we discuss the necessity of ETL like tasks in big data benchmarking and propose the Parallel Data Generation Framework (PDGF) for its data generation. PDGF is a generic data generator that was implemented at the University of Passau and is currently adopted in TPC benchmarks.

1 Introduction

Many big data challenges begin with extraction, transformation and loading (ETL) processes. Raw data is extracted from source systems, for example, from a web site, click streams (e.g. Netflix, Facebook, Google) or sensors (e.g., energy monitoring, application monitoring, traffic monitoring). The first challenge in extracting data is to keep up with the usually very data high production rate. In the transformation step, the data is filtered and normalized. In the last step, data is finally loaded in a system that will then do the processing. This preprocessing is often time-consuming and hinders an on-line processing of the data. Nevertheless, current big data benchmarks, e.g. GraySort [1], YCSB [2], HiBench [3], BigBench [4], mostly concentrate on a single performance aspect rather than giving a holistic view. They neglect the challenges in the initial ETL processes and data movement. A comprehensive big data benchmark should have an end-to-end semantic considering the complete big data pipeline [5]. An abstract example of a big data pipeline as described in [6] is depicted in Figure 1.

Current big data installations are rarely tightly integrated solutions [7]. Thus, a typical big data pipeline often consists of many separate solutions that cover one or more steps of the pipeline. This creates a dilemma for end-to-end benchmarking. Because many separate systems are involved an individual measure for each part's contribution to the overall performance is necessary for making purchase decisions for an entire big data solution. A typical solution to this dilemma

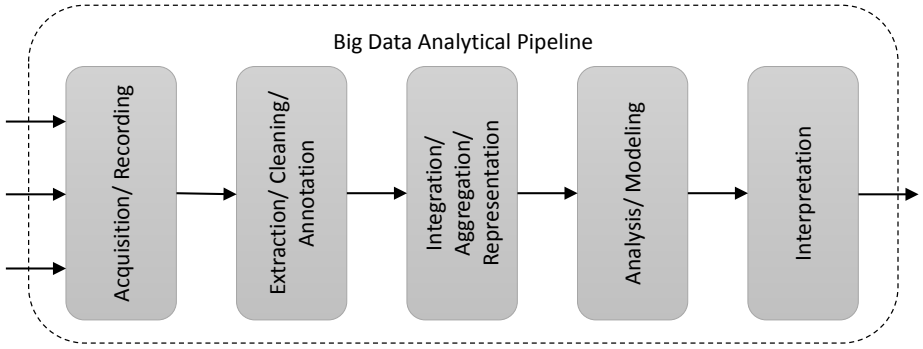

Fig. 1. Abstract Stages of a Big Data Analytics Pipeline

is a component based benchmark. This requires having separate benchmarks for different stages of the big data pipeline. An example is HiBench [3], which includes separate workloads and micro-benchmarks to cover typical Map-Reduce jobs. HiBench, for example, includes workloads for sorting, clustering, and I/O. These micro-benchmarks are run separately and, consequently, inter-stage interactions, i.e., interference and interaction between different stages, as they would appear in real-live systems, are not reflected in the benchmarks.

Considering inter-stage interactions makes the specification of an end-to-end benchmark challenging. This is because it is supposed to be technology agnostic, i.e., it should not enforce a certain implementation of the system under test and also not enforce fixed set of stages. A benchmark should challenge the system as a whole. This creates a dilemma for end-to-end benchmarking of a big data pipeline, because an end-to-end benchmark should not be concerned about the individual steps of the pipeline, which can differ from system to system, but all steps should be stressed during a test. A solution to this dilemma is a benchmark pipeline, where intermediate steps are specified but not enforced and only the initial input and final output are fixed.

For a benchmark to be successful it has to be easy to use. Benchmarks that come with a complete tool chain are used more frequently than benchmarks that consist only of a specification. A recent example is the YCSB, which has gained a lot of attention and a wide acceptance. YCSB is used in many research projects as well as in industry benchmarks (e.g., [8, 9]). For a big data benchmark the most important and challenging tool is the data generator. In order to support the various steps of big data processing, it would be beneficial to have a data generator that can generate the data in different phases consistently. This makes a verification of intermediate results as well as isolate single steps of the benchmark procedure possible and thus further increases the benchmarks applicability. In such a data generator the data properties that are processed (such as dependencies and distributions) need to be strictly computable. A data generation tool that follows this approach is the Parallel Data Generation Framework.

Our major contribution in this article is a solution to the problem of data generation for big data benchmarks with end-to-end semantics. To the best of our knowledge this is the first approach to this problem.

The rest of the paper is structured as follows, in Section 2, we give a brief overview of the Parallel Data Generation Framework. Section 3 describes challenges of big data generation and how they are addressed by the Parallel Data Generation Framework. Section 4 presents related work. We conclude in Section 5 with an outlook on future work.

2 Parallel Data Generation Framework

The Parallel Data Generation Framework (PDGF) is a flexible, generic data generator that can be used to generate large amounts of relational data very fast. It was initially developed at the University of Passau and is currently used in the development of an industry standard ETL benchmark (described in [10]). PDGF exploits parallel random number generation for an independent generation of related values. The underlying approach is straight forward; the random number generator is a hash function which can generate any random number in a sequence in O(1) without having to compute other values. Random number generators with this feature are, for example, XORSHIFT generators [11]. With such random number generators every random number can be computed independently. Based on the random number arbitrary values can generated using mapping functions, dictionary lookups and such. Quickly finding the right random number is possible by using a hierarchical seeding strategy (table → column → row).

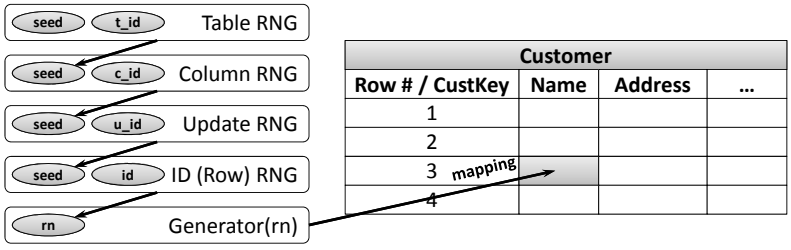

Fig. 2. PDGF's Hierarchical Seeding Strategy

An overview of PDGF's seeding strategy can be seen in Figure 2. The seeding strategy starts by assigning a random number to each table, this number is used as a seed for each column random number generator. PDGF is capable of generating consistent updates, i.e, inserts, deletes, and updates in an abstract time interval (for details refer to [12]). Which and how values are updated is determined by the update random number generator, the resulting seeded row

value random number generator is used to deterministically compute the random numbers required for the actual value generation. Having a seeded random number generator for the value generation instead of a single random number or fixed number of values makes it possible to generate values that use a non-deterministic number of random numbers, such as text.

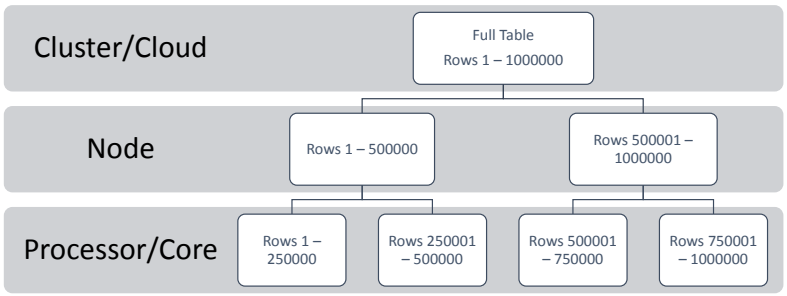

Fig. 3. Parallel Data Generation in PDGF

Not all values should be randomly chosen. An example are references. For tables that contain foreign key constraints, for example, the keys must exist in the referenced tables, which is challenging in the case of non-dense keys or multi-part keys. Using the deterministic approach, existing values can easily and efficiently be recomputed. Furthermore, being able to independently generate all values makes it possible to fully parallelize and distribute the data generation. This especially interesting for big data applications. PDGF comes with an integrated scheduling system that automatically handles multi-core and multi-node parallelism. The working principle is presented in Figure 3. Each table can be split up in equal sized partitions, which can be generated on shared nothing machines. Each partition can further be divided up in multiple subsets, which can be distributed to separate threads or processes.

For further details of this generation approach see [12–17].

3 A Big Data Generator

One can build a versatile data generator for big data benchmarking based on PDGF. Although PDGF was built for relational data it features a post-processing module that enables a mapping to other data formats such as XML, RDF, etc. Since all data is deterministically generated and the generation is always repeatable it is possible to compute intermediate and final results of transformations. The underlying relational model makes it possible to generate consistent queries on the data. This makes PDGF an ideal candidate tool for big data benchmarking.

PDGF was recently used for generating the data set for the BigBench big data analytics benchmark [4]. BigBench models a retail business, were articles

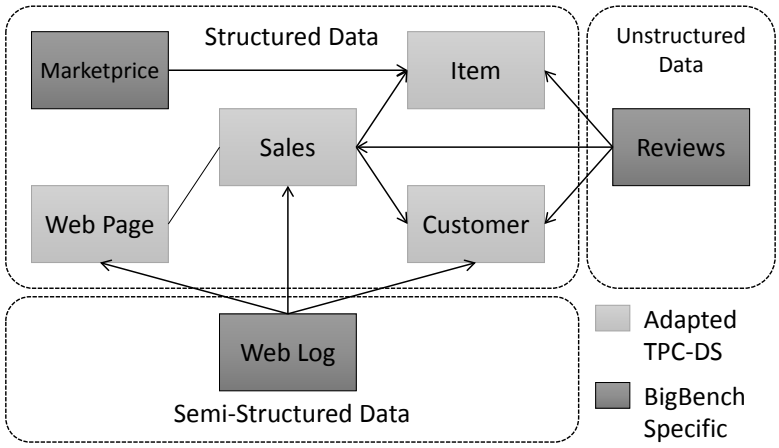

Fig. 4. BigBench Schema

are sold in stores and over websites. The schema consists of structured, semi-structured, and unstructured data as can be seen in Figure 4. The structured core of the schema is adapted from TPC-DS [18]. The semi- and unstructured parts are implemented in PDGF. The unstructured part models reviews of products. The semi-structured part models an Apache Web server log. The reviews are used for sentiment analysis, which requires very realistic text in order to get reasonable results. This is achieved by using Markov chains.

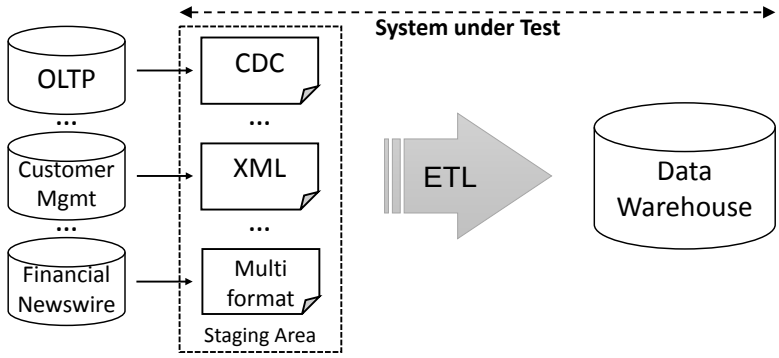

Fig. 5. TPC-DI Overview

Another recently finished data generator built based on PDGF is TPC-DI's data generator. TPC-DI is an data integration benchmark, which benchmarks ETL systems. As is shown in Figure 5, the benchmark defines several sources of information that are stored in different formats. The benchmark itself measures

the performance of a system that integrates the different data sources into a single data warehouse. The data generator generates the historical files of each data source as well *change data captures* in daily increments. For the benchmark to produce meaningful results, the data from the different sources has to be consistent. This means, for example, that only employees with the status account managers in the human resources database manage customers' accounts and, thus, are referenced in the customer management database.

When combining the two examples above, one can create a big data generator that satisfies all characteristics of typical big data use cases. For example, the well established 3 to 5 V's, namely volume, velocity, variety, and the extensions value and veracity, can all be covered by such a generator. Parallel data generation is the only means to generate big volumes of data in timely fashion. The velocity aspect can be satisfied generally by fast generation of data as well as by fast generation of incremental updates, which ensure the characteristic of frequent data change. The variety aspect is covered with different data sources. The value extension is hinting to the additional value that can be retrieved from a deep analysis of the data, which therefore has to have meaningful patterns and correlations. Finally, veracity is of the data can be changed by introducing deltas and errors in the generation, which is present in the TPC-DI specification.

4 Related Work

There are multiple different approaches to data generation. Many current benchmarks use very primitive data that is simply based on statistical distributions. Examples are Terasort (a.k.a. Graysort) [1] and YCSB [2]. In order to get more realistic data, structured approaches to data generation have to be used. One way to get very realistic data is simulation. This can be done in an application specific way, e.g., using human browser interaction simulation with the Selenium simulator [19], or using a generic graph based approach [20, 21]. Although very realistic, all simulation-based approaches are too slow for big data generation. Therefore, many benchmarks including most of the standard benchmarks have special purpose data generators that are not or only to a very small degree configurable. An example are all TPC benchmarks, with the exception of TPC-DI (based on PDGF) and to some extend TPC-DS (based on the partial configurable data generator MUDD [22]). Since the implementation of quality data generators is a tedious work, several commercial and scientific generic data generators have been developed. To ensure fast data generation these typically do not use simulation but either reread data to build correlations (e.g., [23]) or recompute referenced values (e.g., PDGF and Myriad [24]). Due to the data sizes generated and the speed of network and disk transfer rates, the computational approach is the fastest and most scalable and thus most suitable for big data generation.

5 Conclusion

The big data landscape is quickly evolving, much like the landscape of database management systems in its early stages. As a result, big data systems are heterogeneous and even for the broadly accepted Hadoop software stack there is no commonly accepted benchmark. Several proposals are currently emerging, because of the missing maturity of the big data field and the high pace of evolution, benchmarks have to evolve as well. To this end, configurable data generators are necessary to help benchmarks keep up with the development and, thus, stay relevant.

The Parallel Data Generation Framework is an ideal candidate for big data generation. In this article, we have listed characteristics that a big data generator has to fulfill and have demonstrated by example that PDGF can satisfy all requirements. A demo of PDGF is available for download[1]. A commercialized version is available from `http://www.bankmark.de`.

What is missing for an easy to use benchmark is a driver that starts the execution, measures the performance and calculates the metrics. This is non-trivial because there is no standard access language so far. However, relational input as generated by PDGF can be easily transformed in any other representation, which will ease the implementation of such tool chains.

PDGF is continuously improved and extended, current work focuses on data types typical in big data scenarios like text and click-streams. Initial implementations were used to implement the BigBench data generator. Other work targets scaling-up existing data sets and combining simulation-like data generation with the purely computational approach.

References

1. Gray, J.: GraySort Benchmark. Sort Benchmark Home Page – `http://sortbenchmark.org`
2. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking Cloud Serving Systems with YCSB. In: SoCC. (2010) 143–154
3. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis. In: ICDEW. (2010)
4. 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: Proceedings of the ACM SIGMOD Conference. (2013)
5. Baru, C., Bhandarkar, M., Nambiar, R., Poess, M., , Rabl, T.: Benchmarking Big Data Systems and the BigData Top100 List. Big Data **1**(1) (2013) 60–64
6. Baru, C., Bhandarkar, M., Nambiar, R., Poess, M., Rabl, T.: Setting the Direction for Big Data Benchmark Standards. In Nambiar, R., Poess, M., eds.: Selected Topics in Performance Evaluation and Benchmarking. Volume 7755 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 197–208
7. Carey, M.J.: BDMS Performance Evaluation: Practices, Pitfalls, and Possibilities. In Nambiar, R., Poess, M., eds.: Selected Topics in Performance Evaluation and

[1] Parallel Data Generation Framework – `http://www.paralleldatageneration.org`

Benchmarking. Volume 7755 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 108–123
8. Patil, S., Polte, M., Ren, K., Tantisiriroj, W., Xiao, L., Lopez, J., Gibson, G., Fuchs, A., Rinaldi, B.: YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In: SoCC. (2011) 9:1–9:14
9. Rabl, T., Sadoghi, M., Jacobsen, H.A., Gómez-Villamor, S., Muntés-Mulero, V., Mankowskii, S.: Solving Big Data Challenges for Enterprise Application Performance Management. PVLDB **5**(12) (2012) 1724–1735
10. Wyatt, L., Caufield, B., Pol, D.: Principles for an ETL Benchmark. In: TPC TC '09. (2009) 183–198
11. Marsaglia, G.: Xorshift RNGs. Journal Of Statistical Software **8**(14) (2003) 1–6
12. Frank, M., Poess, M., Rabl, T.: Efficient Update Data Generation for DBMS Benchmark. In: ICPE '12. (2012)
13. Poess, M., Rabl, T., Frank, M., Danisch, M.: A PDGF Implementation for TPC-H. In: TPCTC '11. (2011)
14. Rabl, T., Frank, M., Sergieh, H.M., Kosch, H.: A Data Generator for Cloud-Scale Benchmarking. In: TPCTC '10. (2010) 41–56
15. Rabl, T., Lang, A., Hackl, T., Sick, B., Kosch, H.: Generating Shifting Workloads to Benchmark Adaptability in Relational Database Systems. In: TPCTC '09. (2009) 116–131
16. Rabl, T., Poess, M.: Parallel data generation for performance analysis of large, complex RDBMS. In: DBTest '11. (2011) 5
17. Rabl, T., Poess, M., Danisch, M., Jacobsen, H.A.: Rapid Development of Data Generators Using Meta Generators in PDGF. In: DBTest '13: Proceedings of the Sixth International Workshop on Testing Database Systems. (2013)
18. Pöss, M., Nambiar, R.O., Walrath, D.: Why You Should Run TPC-DS: A Workload Analysis. In: VLDB. (2007) 1138–1149
19. Hunt, D., Inman-Semerau, L., May-Pumphrey, M.A., Sussman, N., Grandjean, P., Newhook, P., Suarez-Ordonez, S., Stewart, S., Kumar, T.: Selenium Documentation. (2013) http://docs.seleniumhq.org/docs/.
20. Houkjær, K., Torp, K., Wind, R.: Simple and Realistic Data Generation. In: VLDB '06: Proceedings of the 32nd international conference on Very large data bases, VLDB Endowment (2006) 1243–1246
21. Lin, P.J., Samadi, B., Cipolone, A., Jeske, D.R., Cox, S., Rendón, C., Holt, D., Xiao, R.: Development of a Synthetic Data Set Generator for Building and Testing Information Discovery Systems. In: ITNG '06: Proceedings of the Third International Conference on Information Technology: New Generations, Washington, DC, USA, IEEE Computer Society (2006) 707–712
22. Stephens, J.M., Poess, M.: MUDD: a multi-dimensional data generator. In: WOSP '04: Proceedings of the 4th International Workshop on Software and Performance, New York, NY, USA, ACM (2004) 104–109
23. Bruno, N., Chaudhuri, S.: Flexible Database Generators. In: VLDB '05: Proceedings of the 31st International Conference on Very Large Databases, VLDB Endowment (2005) 1097–1107
24. Alexandrov, A., Tzoumas, K., Markl, V.: Myriad: Scalable and Expressive Data Generation. In: VLDB'12. (2012)