BigBench: Towards an Industry Standard Benchmark for Big Data Analytics

Ahmad Ghazal[1,5], Tilmann Rabl[2,6], Minqing Hu[1,5],
Francois Raab[4,8], Meikel Poess[3,7], Alain Crolotte[1,5], Hans-Arno Jacobsen[2,9]

[1]Teradata Corp., [2]University of Toronto, [3]Oracle Corp., [4]InfoSizing, Inc.
[5]{ahmad.ghazal,minqing.hu,alain.crolotte}@teradata.com, [6]tilmann@msrg.utoronto.ca
[7]meikel.poess@oracle.com, [8]francois@sizing.com, [9]jacobsen@eecg.toronto.edu

ABSTRACT

There is a tremendous interest in big data by academia, industry and a large user base. Several commercial and open source providers unleashed a variety of products to support big data storage and processing. As these products mature, there is a need to evaluate and compare the performance of these systems.

In this paper, we present BigBench, an end-to-end big data benchmark proposal. The underlying business model of BigBench is a product retailer. The proposal covers a data model and synthetic data generator that addresses the variety, velocity and volume aspects of big data systems containing structured, semi-structured and unstructured data. The structured part of the BigBench data model is adopted from the TPC-DS benchmark, which is enriched with semi-structured and unstructured data components. The semi-structured part captures registered and guest user clicks on the retailer's website. The unstructured data captures product reviews submitted online. The data generator designed for BigBench provides scalable volumes of raw data based on a scale factor. The BigBench workload is designed around a set of queries against the data model. From a business prospective, the queries cover the different categories of big data analytics proposed by McKinsey. From a technical prospective, the queries are designed to span three different dimensions based on data sources, query processing types and analytic techniques.

We illustrate the feasibility of BigBench by implementing it on the Teradata Aster Database. The test includes generating and loading a 200 Gigabyte BigBench data set and testing the workload by executing the BigBench queries (written using Teradata Aster SQL-MR) and reporting their response times.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance measures*

Keywords

Benchmarking; big data; map reduce

1. INTRODUCTION

Today's data explosion, fueled by emerging applications, such as social networking, micro blogs, and the "crowd intelligence" capabilities of many sites, has led to the "big data" phenomenon. It is characterized by increasing volumes of data of disparate types (i.e., structured, semi-structured and unstructured) from sources that generate new data at a high rate (e.g., click streams captured in web server logs). This wealth of data provides numerous new analytic and business intelligence opportunities like fraud detection, customer profiling, and churn and customer loyalty analysis.

Consequently, there is tremendous interest in academia and industry to address the challenges in storing, accessing and analyzing this data. Several commercial and open source providers already unleashed a variety of products to support big data storage and processing. These tools are mostly parallel database management systems (e.g., Greenplum[4], Netezza's TwinFin[9], Teradata[8], Oracle[6]) or MapReduce (MR) based systems (e.g., Hadoop [1], Cloudera's CDH [3], Hive[2] and many other systems like those in [15, 17, 24, 27]).

As big data systems mature, the pressure to evaluate and compare performance and price performance of these systems rises. However, to date there are no standard benchmarks available. This takes us back to the middle of the 1980's, when the lack of standard database benchmarks led many database management system vendors to practice what is now referred to as "benchmarketing" – a practice in which organizations make performance claims based on self-defined, highly biased benchmarks. The goal of publishing results from such tailored benchmarks was to state marketing claims, regardless of the absence of relevant and verifiable technical merit. In essence, these benchmarks were designed as forgone conclusions to fit a pre-established marketing message. Similarly, vendors would create configurations, referred to as "benchmark specials", that were specifically designed to maximize performance against a specific benchmark with limited benefit to real-world applications.

Towards the end of the 1980's, as a response to this growing practice, benchmark consortia such as the Transaction Processing Performance Council (TPC) and the Standard Performance Corporation (SPEC) were founded. Influenced by academic database experts and well-known industry lead-

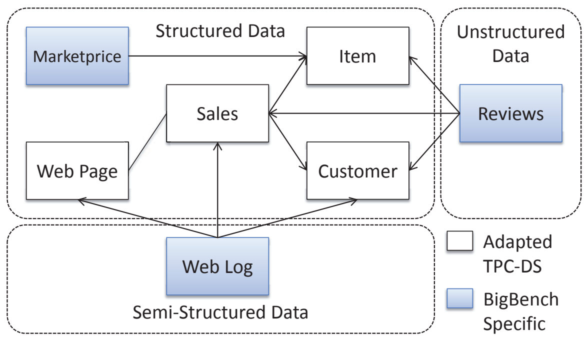

Figure 1: Big Data Benchmark Data Model

ers, industry standard benchmarks such as TPC-A, TPC-C and TPC-D were engineered and rules around publishing results were agreed upon.

Recently a few efforts in the area of big data benchmarks emerged, such as YCSB[16], PigMix[7], GridMix [5] and GraySort [20]. These efforts are island solutions and not policed by any industry consortia. While some are focused on one or a subset of components and tasks typical for big data systems, others are based on specific map-reduce-style systems.

We believe an industry standard big data benchmark must be an end-to-end benchmark covering all major characteristics in the lifecycle of a big data system including the three Vs described by Douglas Laney[21]: (i) *volume* (larger data set sizes), (ii) *velocity* (higher data arrival rates, such as click streams) and (iii) *variety* (increased data type disparity, such as structured data from relational tables, semi-structured data from key-value web clicks and un-structured data from social media content).

In this paper, we present our proposal for an end-to-end big data benchmark. After a presentation of initial ideas for the benchmark at the first Workshop on Big Data Benchmarking[1] a group formed that collaborated on building the specification. We call it "BigBench". It is based on a fictitious retailer who sells products to customers via physical and online stores. The proposal covers a data model, synthetic data generator and workload description. The workload queries are specified in English, since no clear standard for big data systems has yet emerged. We also suggest directions for big data metrics specific to data loading and workload execution. Furthermore, the feasibility of the proposal is validated by implementing it on the Teradata Aster DBMS (TAD). This experiment involves generating 200 Gigabyte of raw data and loading it into TAD. The English like workload queries are implemented using TAD's SQL-MR syntax and executed as a single stream of queries.

The first major component of BigBench is the specification of a data model that focuses on volume, variety and velocity. The variety property of our model is illustrated in Figure 1. The structured part of BigBench is adapted from the TPC-DS data model, which also depicts a product retailer [23]. We borrowed the store and online sales portion from that model and added a table for prices from the retailer's competitors.

The structured part is enriched with semi-structured and un-structured data shown in the lower and right hand side of Figure 1. The semi-structured part is composed by clicks made by customers and guest users visiting the retailer's web site. Our design assumes the semi-structured data to be in a key-value format similar to Apache's web server log format. The un-structured data in our model is covered by product reviews that can be submitted by guest users or actual customers.

We also provide the design and implementation of a data generator for the proposed BigBench data model. Our data generator is based on an extension of PDGF [29]. PDGF is a parallel data generator that is capable of producing large amounts data for an arbitrary schema. The existing PDGF can be used to generate the structured part of the BigBench model. However, it is not capable of producing neither the semi-structured web clicks nor the unstructured product reviews text. Part of our contribution in this paper is to extend PDGF to cover the semi-structured and un-structured parts. We enhanced PDGF to produce a key-value data set for a fixed set of required and optional keys. This is sufficient to generate the web logs part of BigBench.

The main challenge in generating product reviews is to produce un-structured text. We developed and implemented an algorithm that produces synthetic text based on some sample input text. The algorithm uses a Markov Chain technique that extracts key words and builds a dictionary based on these key words. The new algorithm, called TextGen, is applied or our retailer model by using some real product reviews from amazon.com for the initial sample data. PDGF interacts with TextGen through an API sending product category as input and getting a product review text for that category.

The volume dimension of our model is far simpler than the variety discussion and previous data generators had a good handle on that. PDGF handles the volume well since it can scale the size of the data based on a scale factor. It also runs efficiently for large scale factors since it runs in parallel and can leverage large systems dedicated for the benchmark. We also address big data velocity by establishing a periodic refresh scheme that constantly adds data to the different areas of the data model.

The second major component of BigBench is the specification of workload queries applied on the BigBench data model. In terms of business questions, we found that the big data retail analytics by McKinsey [22] serves our purpose given that BigBench is about retail. In [22] five major areas, or business levers, of big data analytics are identified: marketing, merchandising, operations, supply chain and new business models.

In addition to the big data retail business levers above, we looked at three different technical dimensions the BigBench queries should span. The first technical dimension is about the type of data used in queries. This implies making sure that structured types, semi-structured types, unstructured types and their combinations are each covered in the queries. The second technical dimension covers the two common paradigms of declarative processing (SQL and similar constructs like HQL) and procedural MR processing. To that end, some queries are best suited to be declarative, others to be procedural and others to be a mix of both. The third technical dimension is about the different algorithms of analytic processing as described by the Apache MAHOUT system. Examples of these algorithms are classifications,

[1]WBDB, May 2012, San Jose – `http://clds.ucsd.edu/wbdb2012`

pattern matching, clustering, regression, dimensional reduction, etc.

In summary, our key contributions are as follows:

1. We present the first end-to-end benchmark for big data analytics while previous work focused on few selected types of data or processing. BigBench implements the complete use-case of a realistic retail business.

2. We specify 30 queries that cover all important aspects of big data analytics. The queries are specified in English as well as TAD's SQL-MR syntax.

3. We develop and implement a novel technique for producing un-structured text data and integrate it with a traditional structured data generator.

4. We conduct a proof of concept implementation and evaluation of BigBench by executing the benchmark on the Teradata Aster DBMS.

The remainder of this paper is organized as follows. Section 2 covers previous work related to big data benchmarking. Section 3 gives a detailed description of the BigBench benchmark. The data model and data generation are described in detail in Sections 3.1 and 3.2. We describe the workload queries in Section 3.3 and the benchmark metrics in Section 3.4. We present our proof of concept implementation of BigBench using TAD in Section 4 including results involving 200 Gigabyte database. Finally, Section 5 summarizes the paper and suggests future directions.

2. RELATED WORK

The requirement for well defined benchmarks that measure the performance of DBMS dealing with very large amounts of data emerged when the first generation of commercial systems appeared in the 1980's by Teradata Corporation and other more traditional DBMS vendors, who followed. Driven by vendor's needs to compare commercial systems, the Transaction Processing Performance Council developed a series of data warehouse end-to-end benchmarks starting with TPC-D in the beginning of the 90's and TPC-H and TPC-R in the dawn of 2000 (all specifications available from the TPC website[2]). These benchmarks, restricted to terabyte data sizes, emphasized single and multi-user performance of complex SQL query processing capabilities with some updates on an enterprise data warehouse. Even earlier, academia started developing micro benchmarks such as the Wisconsin benchmark, the OO7 [12] and BUCKY [13] benchmarks for object-oriented DBMSs, XMark [31] and EXRT [14] benchmarks for XML-related DBMS technologies.

As data volumes grew from megabytes of data and simple data models (small number of tables with few relationships) over time to petabytes and complex data models (large number of tables with many complex relationships) the TPC responded with the development of its next generation decision support benchmark, TPC-DS [23], in the early 2000's. Still based on the SQL programming language it contains many big data elements, such as very large data and system sizes. Although the current limit is 100 terabyte, the data generator and schema can be extended to petabytes. It also contains very complex analytical queries using sophisticated SQL structures and a concurrent update model.

In parallel, academia as well as emerging big data companies have started defining the next generation big data benchmarks, which are mostly component and micro benchmarks. Yahoo! developed its cloud serving benchmark, YCSB, to evaluate NoSQL data stores [16]. It is a flexible multiuser benchmark with two tiers, a performance tier (testing latency) and a scalability tier. In the original paper, three workloads were runagainst four different data stores: HBase, Cassandra, PNUTs, and MySQL. Other evaluations followed that extended the scope of YCSB [30, 25]. The CALDA effort [26] defined a micro-benchmark for big data analytics based on Google's MapReduce paper and compared Hadoop with two RDBMS systems, one that is row and one that is column organized. Another widely used benchmark is the TeraSort or GraySort benchmark [20], which can be considered a micro benchmark that sorts a large number of 100-byte records doing considerable amount of computation, networking, and storage I/O. Other benchmarks are the GridMix [5] and PigMix [7].

TPC-DS [23, 28] is TPC's latest decision support benchmark. It covers the major three disciplines in the life-cycle of a relational decision support benchmark, namely (i) loading the initial database (ii) executing queries in both single- and multi-user modes (iii) refreshing the database. TPC-DS handles some aspects of big data like volume and some aspects of velocity. Still, it lacks key components of big data like semi-structured and unstructured data and their associated analytics.

In summary, previous benchmarks described in this section are mostly micro and component benchmarks. Others like TPC-DS lack key big data characteristics. This brings a need for an end-to-end benchmark for big data processing.

3. BIG DATA BENCHMARK

This section covers the major parts of the BigBench specification. Due to space restrictions, not all details can be presented here. Additional details can be found in an extended version of this paper, to be made available at publication time.

3.1 Data Model

The three cornerstone aspects of big data systems are *volume*, *variety*, *velocity*. Big data systems need to be able to deal with large volumes of data, sometimes in the multiple petabyte range. We deal with the volume aspect in the following section about data scaling. Variety refers to the ability to deal with differently organized data, from unstructured to semi-structured and structured data. The following section about variety lays out a structure that covers all the types of data integrated in one model. Velocity refers to the ability of a big data system to stay current through periodic refreshes, commonly referred to as *extraction*, *transformation* and *load* (ETL). A big data system is not a one-time snapshot of a business operations database nor is it a database where OLTP applications are running concurrently. Hence, staying current with the operational side is a very important aspect of analytical systems, and even more so in the context of a big data system.

In the following subsection, we develop the data model showing how the 3 Vs in big data are addressed in BigBench. We show how *volume* is addressed by using scale

[2]TPC - `http://www.tpc.org`

```
127.0.0.1 - - [Jun/23/2003:05:59:23 +0200]
"GET/page33.html?wcs_click_date=2452814
&wcs_click_ time=21563&wcs_user_id=95789
&wcs_web_page_sk=32&wcs_item_sk=28 HTTP/1.1" 200 2256
"http://www.someurl.org" "Mozilla/5.0"
```

Figure 2: Example of a web log entry

factors in the data generators to scale data up to petabytes of data, how *variety* is addressed through the usage of data from many sources and how *velocity* is achieved by periodic refreshes of the data repository.

3.1.1 Variety

The general benchmark data model is summarized in Figure 1, which shows the three data components of the benchmark namely structured data, semi-structured data and unstructured data together with the relationships between them.

The structured component of BigBench is adapted from the TPC-DS benchmark recently published by the TPC [10]. A description of this benchmark can be found in [23, 28]. BigBench is however not a simple extension of TPC-DS. Instead, BigBench focuses chiefly on the analytics associated with semi-structured and unstructured data.

With a few exceptions most of the tables contained in TPC-DS are used by BigBench; the main focus being store and web sales, which only contain structured data. These tables cover data relating to the purchases made in stores and over the web, but also related tables such as *item* describing the items offered by the retailer, *customer* and its ancillary tables containing all relevant client data, *web_page* and *web_site* describing pages and web sites used by online clients and all associated dimension tables. To better support our functional design, we also added a new table called *item_marketprices* to the structured data. It contains competitor names and prices for each item so that price comparisons performed by online users who are interested in particular items could also be captured.

The semi-structured data focuses on click-streams, contained in web log files. While some of the clicks result in sales thereby necessitating a link to structured area tables containing online sales, item, web pages, customer and associated dimensions, the large majority of these clicks are associated with browsing activity not resulting in sales. These clicks focus on items and are associated with registered users or guests.The format retained for the clicks is that of Apache logs. A typical entry of such a log associated with a registered user could look like the example in Figure 2.

Web logs can be processed either directly at run time (late binding) or parsed and stored into a structured table/file. Since all values are surrogate keys referring to the structured schema, the above record once processed could look like Table 1.

The unstructured data resembles written text associated with product reviews of items offered by the retailer. Such reviews could be from several sources, namely guest users, registered users with a purchase and registered users without a purchase. This implies a relationship between reviews and structred data like customer, sales and item tables. The reviews and its relationship with the structred data can be captured by a table/file. The table/file captures the primary keys of the referenced tables. The review itself is contained in a large variable character field containing free form text, the rating score and the date and time of the review are also contained in the table/file.

3.1.2 Volume

The size of the structured area is based on the size of the tables involved, using a well-understood and known quantity similar to the scale factor in TPC-DS. The size of the semi-structured and unstructured areas are also based on this scale factor. Consequently, the size of the complete BigBench data set is based on a single scale factor and is predictable and deterministic at any volume.

For the item_marketprice table, it is assumed that an average of 5 competitor prices are stored for each item. Thus, the sizing of item_marketprice is $|\text{item}| \times 5$.

The size of web logs depends on the number of clicks made by buyers (making entries in web_sales) and visitors who do not end up buying. Each row in web_sales represents a single line item, thus the number of clicks per sale is comprised of the number of clicks per item and the number of clicks to make a sale (i.e. login, go to cart, checkout). The number of clicks for buyers c_b can be specified with the following equation:

$$c_b = |\text{web\_sales}| \times (\text{pages per item} + \frac{\text{pages per buy}}{\text{items per sale}})$$

Assuming both pages per item and pages per buy to be equal to 4 on average and setting the avergae value of items per sale to be 12 (from TPC-DS), the value of c_b is simplified to

$$c_b = |\text{web\_sales}| \times 4.33$$

We assume that 80% of surfers are visitors (20% buyers) which makes the ratio of visitors to buyers to be 4:1. We also assume that on average visitors browse items the same way as buyers. Based on these assumptions, the formula for the number of clicks for visitors c_v is:

$$c_v = (|\text{web\_sales}| \times \text{pages per item}) \times \text{visitor ratio}$$
$$c_v = |\text{web\_sales}| \times 16$$

Overall, the size of the web log is $c_b + c_v$ and can be expressed as a multiple of the size of web_sales. It is web_sales$\times$ 20.33. The web_sales table scales linearly with the scale factor, the size for scale factor 1 is 720K, thus the number of entries for the web log at scale factor 1 is 14,600K. Given to the log format, the raw file size is 3 gigabyte.

For the review sizing, a similar approach is chosen. Three sources for reviews are considered: anonymous reviews, random item reviews by registered users (customers), and reviews based on sales. The number of anonymous reviews is related to the number of items, an average of 5 anonymous

Table 1: Representation of a web log entry

Field Name	Value
wcs_click_sk	996146
wcs_click_date_sk	2452814
wcs_click_time_sk	21563
wcs_item_sk	28
wcs_web_page_sk	32
wcs_user_sk	95789

reviews per item is assumed. The number of reviews by registered users is dependent on the number of users in the system. Because not all users are actually writing reviews, an average of one review per 5 users is assumed. Finally, a certain amount of the sales will directly lead to a review. This amount is set to 15%. The number of reviews can be computed by the following formula:

$$|\text{reviews}| = |\text{items}| \times 5 + |\text{customers}| \times 0.2 + |\text{web\_sales}| \times 0.15$$

3.1.3 Velocity

Velocity, i.e. a periodic data refresh process, is an integral part of the life cycle of a big data system. A production data refresh process consists of three steps: (i) data extract (ii) data transformation, and (iii) data load . In a production system environment, the data extraction step may consist of numerous separate extract operations, executed against multiple operational systems and ancillary data sources. As it is unlikely that the full list of these operational data sources resides on the system running the big data application, it is doubtful the measurement of the data extraction performance would result in a metric appropriate or meaningful to the scope of this benchmark. In light of this, the data extract step is assumed and represented in the benchmark in the form of generated files.

There are two aspects to discuss in a periodic refresh model for the tables in BigBench: (i) amount of data to include in the refresh process and (ii) the time interval at which the refresh occurs. Both aspects apply to the structured (web sales channel and *item_marketprice* tables), semi-structured (*click_stream*) and un-structured data (*product_review*).

We implement BigBench's periodic refresh process based on the well studied methodology for data maintenance of TPC-DS. It defines the insertion of new data and the deletion of old data from all fact tables as well as insert and updated data of dimensions. Dimensions are divided into three sets, history keeping, non-history keeping and static dimensions. Static dimensions, such as date and time are not updated. History keeping dimensions never overwrite any data, but they keep a history of all former changes. Non-History keeping dimensions resemble almost a one-to-one copy of the table in the operational system of the business, i.e. they update existing data. Both, history keeping and non-history keeping dimensions, accept new data and never delete any old data. According to the above definitions, *click_stream* and *product_review* are fact tables and *item_marketprice* is a history keeping table. Pseudo code for the insertion, deletion of fact table data as well as insert and update operations for the dimension tables can be found in [23] and the official TPC-DS specification [3].

One of the fundamental aspects of the above methodology is the concurrent execution of the refresh process with the query workload. Queries must be interspersed with insert, delete and update operations. In BigBench we run N concurrent query streams containing queries against the structured, semi-structured and unstructured portions of the schema. The number of refresh processes executed is a linear function of the number of query streams, S. In real systems, data against the different data portions is updated with different frequencies. Hence we define a vector V with the following three separate data refresh velocities for each of

[3]TPC – `http://www.tpc.org`

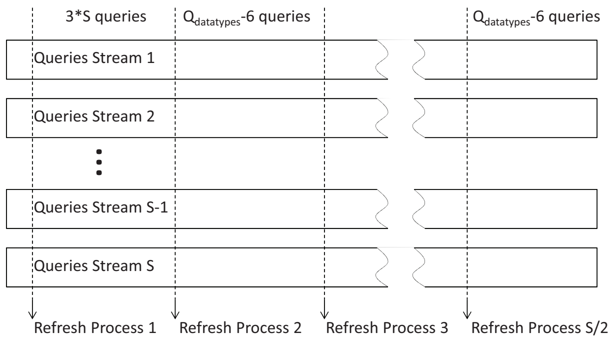

Figure 3: Scheduling of refresh processes based on executed queries per data type

the different data portions, $V = (V_{structured}, V_{semistructured}$ and $V_{unstructured})$. We suggest the following values for V, which are subject to change as we run more experiments. The structured data being the least frequently updated portion of the schema has a velocity of $V_{structured} = 1$, i.e. S refresh process. The unstructured data gets a velocity of $V_{unstructured} = 2 * V_{structured}$, i.e. $2 * S$ refresh process, and the semi-structured data being the most frequently updated portion gets a velocity of $V_{semistructured} = 2 * V_{unstructured}$, i.e. $4 * S$ refresh process. The total number of refresh processes is $7 * S$.

During a BigBench run the following two requirements guarantee that the queries are interspersed with the queries (S is the total number of query streams and Q_{data_type} is the total number of queries against the three portions of the schema):

1. The Nth refresh set can only start after $[((3*S)+((N-1)*2*Q_{data_type})]$ queries have completed (aggregated over all streams), and

2. The $[(3*S)+(N*(Q_{data_type}-6))+1]$th query (aggregated over all streams) can only start after the Nth refresh set has completed.

This means that at least $(3*S)$ queries must complete before the first refresh set can start and at least $Q_{data_type} - 6$ additional queries must complete before the second refresh set can start. In general at least $(3*S)+((N-1)*Q_{data_type}-6))$ queries must complete before the Nth refresh set can start. Figure 3 shows how the refresh processes are scheduled depending on the number of executed queries.

All three type of data tables follow the well-understood scale factors of TPC-DS as outlined in the previous section. That is the amount of data to be inserted in each ETL operation is a percentage of the initial load, e.g. 0.1%.

3.2 Data Generation

Our data generation design is based on an existing technology called Parallel Data Generation Framework (PDGF). PDGF was designed to address structured data. Part of the work presented in this paper is to extend the framework to produce the semi-structured and unstructured data. The semi-structured data is generated in form of weblogs and the unstructured data in form of item reviews. In the following section, we give an overview of PDGF and then elaborate on its extensions for semi-structured and unstructured data.

3.2.1 PDGF

PDGF is a generic, parallel data generator which was developed at the University of Passau [18, 29]. PDGF is implemented in Java and fully platform independent. Currently, PDGF is used to implement the default data generator for the TPC's new ETL benchmark TPC-DI [33]. PDGF's generation approach exploits the inherent parallelism of xorshift random number generators by using a novel seeding strategy. The seeding strategy hierarchically assigns seeds to the tables, columns and rows of a database schema and thus makes it possible to generate data completely in parallel as well as re-calculate any value in the database without accessing the original data.

Originally, PDGF is designed to generate relational data. The data is specified in two XML documents, the schema configuration and the generation configuration. As the name suggests, the schema configuration specifies the data similar to the definition of a relational schema. The generation configuration makes it possible to specify additional post-processing of the generation. The post-processing includes formatting data, merging and splitting tables, as well as advanced procedures by providing a script like programming interface using the Javassist[4] library.

PDGF can be used *as is* to generate the structured parts of the data model. As discussed above, the current BigBench schema comprises three additional entities on top of the TPC-DS schema: the Item_marketprice table, an apache-style web server log, and the online reviews. The Item_marketprice table is a regular table and can easily be generated using PDGF. In Listing 1, an excerpt of the specification of Item_marketprice can be seen. The table is defined in a way similar to the SQL definition language, with an additional specification of the generation rules. The surrogate key (imp_sk) is, for example, generated with a ID generator. PDGF supports more complex generation specifications as can be seen in the case of the imp_competitor field, this field is generated as a random string that is *null* with a probability of 0.025%.

```
<property name="Item_marketprice" type="double">
  ${item}*${avg_competitors_per_item}
</property>

<table name="Item_marketprice">
  <size>${Item_marketprice}</size>
  <field name="imp_sk" size="" type="NUMERIC">
   <gen_IdGenerator/>
  </field>
[..]
  <field name="imp_competitor" size="20"
       type="VARCHAR">
   <gen_NullGenerator>
    <probability>0.00025</probability>
    <gen_RandomAString>
     <size>20</size>
    </gen_RandomAString>
   </gen_NullGenerator>
  </field>
[..]
</table>
```

Listing 1: Excerpt of the Schema Definition for Item_marketprice

The web server log has a special formatting, an example is shown in Figure 2. To generate a realistic web log, we specified a table in PDGF that has all required columns for a web log entry and formated it using PDGF's scripting capabilities. Below in Listing 2 an excerpt of the definition of the web server log table can be seen. The excerpt shows the definition of the size of the web log, and the table definition with two attributes. The sizing is computed according to the formula in Section 3.1.2, the specification of the parameters of the formula is omitted. For the table itself only two attributes are shown: a surrogate key *wcs_click_sk* and the reference to the web page *wcs_web_page_sk*. This reference is null with a probability of 0.00025. In Listing 3, the formatting code for the web log can be seen. As shown in the listing, some of the values in the log are static. For example the request IP address is always "127.0.0.1" while other values such as the time and date are extracted from the table.

```
<property name="Web_clickstreams" type="double">
  (${sales} * (${pages_per_item} + (${pages_to_buy}
       / ${items_per_cart}))
  + (${sales} * ${buy_ratio} * ${pages_per_item})
</property>
<table name="Web_clickstreams">
  <size>${Web_clickstreams}</size>
  <field name="wcs_click_sk" size="" type="NUMERIC">
    <gen_IdGenerator/>
  </field>
[..]
  <field name="wcs_web_page_sk" size=""
       type="NUMERIC">
    <gen_NullGenerator>
      <probability>0.00025</probability>
      <gen_LongGenerator>
        <min>1</min>
        <max>${web_page}</max>
      </gen_LongGenerator>
    </gen_NullGenerator>
  </field>
[..]
</table>
```

Listing 2: Excerpt of the web log specification

```
<output name="CompiledTemplateOutput" >
  <template><!--
String nl    =
     pdgf.util.Constants.DEFAULT_LINESEPARATOR;
buffer.append("127.0.0.1 - - [" + fields[4] + ":" +
     fields[5] + " +0200] ");
buffer.append("\"GET /page"+fields[7]+".html?");
[..]
buffer.append(" HTTP/1.1\" 200 0 - \""+fields[1]);
buffer.append("\" \"Mozilla/5.0 \""+ nl);
  --></template>
</output>
```

Listing 3: Excerpt of the formatting instructions for the web log

The review generator was built as a standalone program, it is configured using an XML document that specifies all parameters for each review. In order to generate reviews that correlate with the structured data, e.g. the items that are reviewed exist in the database and the registered reviewers are actual customers, PDGF is used to generate the XML configuration for the review generator. This is also done using the scripting interface. Again, a table is specified in PDGF that contains all required information and the rows

[4]Javassist project homepage - http://www.csg.is.titech.ac.jp/~chiba/javassist/

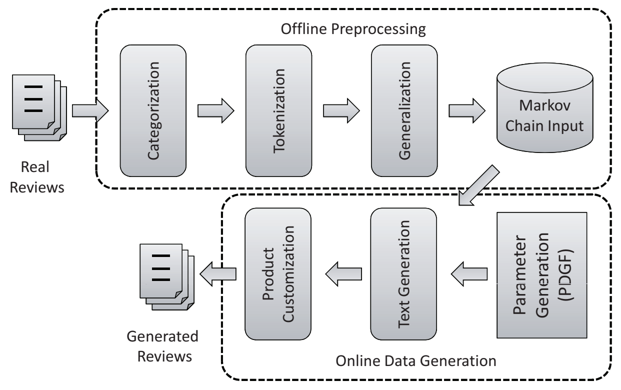

Figure 4: Review Generation Process

are output as XML document fragments. Details on the review generation are given in the section below.

3.2.2 Review Generation

Reviews build the unstructured part of our data set. They are an integral part of the data model and have to be processed. Thus they need to contain realistic and useful information. As discussed below in the workload section, the benchmark contains queries that require sentiment analysis and similar text analysis on the reviews. We have developed a novel approach for generating the reviews that is based on text generation using Markov chains [11].

In Figure 4 an overview of the review generation process can be seen. The process can be separated in two phases. An offline phase, that processes real reviews and generates a knowledge base for the review generation and an online phase that generates reviews based on the knowledge base.

The offline process starts with collecting real reviews from online resources. For our proof of concept, we collected a set of 150 reviews per category from an online retailer. In the first processing step the reviews are categorized by product type. For the categorization, we use an intersection of product categories from the online retailer and the class and category hierarchy in the item dimension in the TPC-DS schema. The online reviews have a rating which is used to create an orthogonal categorization for the review sentiment. The crawler also collects statistical information about the number of reviews per item, the length of reviews and the distribution of ratings. Since reviews are tailored to a specific product, they are tokenized and the review subject is generalized. For now this process only includes filtering out product names and replacing them with generic identifiers. Although this approach removes the product name from reviews, they are still highly domain specific. Since the generalization is an offline process that has to be done only once, the computation can be more involved. In future versions of the generator more sophisticated approaches will be implemented.

Using the tokenized and generalized reviews, the transition probabilities between words in the text are analyzed and stored. These probabilities are know as Markov chains. An *order-1* chain will only store the frequency of a word appearing after another one. So for each word all possible successors and the frequency in which they appear is stored. To get more realistic text, more than one predecessor can be taken into account for generating the text. In practice, we use order-2 to order-4 text to achieve high quality reviews. An excerpt of an order-2 generated text can be seen below.

> My review title says it all. I wanted to like it, because it's a good subject. Didn't flow well, some times confusing. This book is not a self help book, this may be worth reading for that alone.

The review generator was implemented as a standalone program that is configured by an XML document. The configuration contains one `<review>` element for each review that should be generated. For each review the item ID, category, user name, transaction ID, date, time, rating and word count are specified. This information is generated by PDGF and later fed to the review generator. This way, it is assured that the review data is consistent with the data generated by the other generators. In future revisions of the benchmark, all parts of the data generation will be implemented within PDGF.

3.3 Workload

In this section, we present the proposed workload for BigBench. In addition to the queries described below, we consider the initial database population as part of the workload . We refer to this initial phase as transformation ingest (TI). TI covers the ETL process, including any steps needed to prepare the data before querying (e.g., indexing or statistics collection).

The main part of the workload is the set of queries to be executed against the data model. These queries are designed along one business dimension and three technical dimensions, aiming to cover different business cases and technical perspectives. Our business cases are based on Mckinsey's report on big data [22]. From a technical perspective, we focus on data sources, processing types and analytical techniques.

Following the approach used for most TPC benchmarks, The BigBench queries are defined in terms of business questions and expressed in plain English. We created a total of 30 business questions for the BigBench workload. Note that, due to the limited space, we do not present all of the 30 queries in this paper. The complete set of BigBench queries can be found in an extended version of this paper. In addition to the English definition of the queries, we also present them using Teradata Aster's SQL-MR syntax [19, 32].

The remainder of this section is organized as follows: first, we discuss the business cases with query examples. We then present the three technical dimensions and show the distribution of queries along each of the dimensions.

3.3.1 Business Cases

The McKinsey report gives a comprehensive view of big data's transformative potentials for retail business. From the report, we identified nine big data retail levers that fit in the BigBench workload. Furthermore, we added return analysis under the category Operations which makes a total of ten levers. (Returns are often connected with frauds, which makes it important from a business perspective.) These ten levers fall into the following five main categories: Marketing, Merchandising, Operations, Supply Chain and New Business Models. The organization of the ten levers into these five categories is shown in Table 2.

In the following, we present the ten retail levers and we illustrate each lever with a sample query.

Table 2: Levers Within Business Categories

Business category	Big data lever
Marketing	-Cross-selling -Customer micro-segmentation -Sentiment analysis -Enhancing multichannel consumer experience
Merchandising	-Assortment optimization -Pricing optimization
Operations	-Performance transparency -Return analysis
Supply chain	-Inventory management
New business models	-Price comparison

1. **Cross-selling:** In this lever, we include queries involving market basket analysis and collaborative filtering based recommendations. For example, Query 1 computes the probability of browsing products from a category after customers viewed items from another category.

   Query 1: Perform category affinity analysis for products purchased online together.

2. **Customer micro-segmentation:** Queries in this lever ranges from grouping users using one dimension to clustering users using more sophisticated features. Query 2 tries to cluster users into eight groups based on their purchase history.

   Query 2: Customers are separated along the following key shopping dimensions: recency of last visit, frequency of visits and monetary amount. Use the in-store and online purchase data over a calendar year to compute.

3. **Sentiment analysis:** These queries involve an enormous amount of text and natural language processing, including detecting sentiment words or phrases from reviews, determining sentiment polarity, etc., as shown in Query 3.

   Query 3: For a given product, extract sentences from its product reviews that contain sentiments and display their sentiment polarity.

4. **Enhancing multi-channel consumer experience:** Queries in this lever are targeted at understanding users shopping behaviors through both online and in-store channels. Query 4 checks if online browsing affects customers' in-store purchase behaviors by measuring the number of days between the two activities.

   Query 4: Find all customers who viewed items of a given category on the web site in a given month and year and subsequently made an in-store purchase in the same category within the following three months.

5. **Assortment optimization:** In this lever we focus on queries that identify products, categories or stores that can be targeted for improvements. Query 5 finds the products with decreasing sales.

   Query 5: Find the categories with flat or declining sales for in-store purchases during a given year for a given store.

6. **Pricing optimization:** Queries in this lever are focused on measuring the impact of price changes on sales, as shown in Query 6.

   Query 6: Compute the impact on sales of an item price change by computing the total sales for items in a 30-day period before and after the price change. Group the total sales by items and location of warehouse where they were delivered from.

7. **Performance transparency:** Our queries for this lever are about finding stores with downward or upward performance. Query 7 identifies stores with downward sales and finds possible reasons through available reviews.

   Query 7: Identify stores with flat or declining sales in 3 consecutive months, check if there are any negative online reviews regarding these stores.

8. **Return analysis:** These queries target two areas; identifying problematic products and detecting refund fraud. Query 8 first finds products with high return rate and then identifies if there are any issues from product reviews.

   Query 8: Retrieve the items with the highest number of returns where the number of returns was approximately equivalent across all stores and web channels (within a tolerance of +/- 10%), within a week ending a given date. Analyze the online reviews for these items to see if there are any major negative reviews.

9. **Inventory management:** Queries for this lever focus on statistical analysis on product inventory. Query 9 computes the mean and variation of item inventories and identifies those with large variations.

   Query 9: This query contains multiple, related iterations. Iteration 1 calculates the coefficient of variation and mean of inventory by item and warehouse for two consecutive months. Iteration 2 finds items that had a coefficient of variation in the first months of 1.5 or larger.

10. **Price comparison:** In this lever, we have one query that measures the correlations between competitor's prices and item sales, as shown in Query 10.

    Query 10: For a given product, measure the effect of competitor's prices on products' in-store and online sales.

The business cases were the main driver for the definition of the BigBench queries. The bulk of the queries are within the Marketing and Merchandising categories since these two are the most commonly used and can be further divided in sub-categories, as discussed in [22]. The overall breakdown of queries over the five business categories is shown in Table 3.

3.3.2 Technical Dimensions

In the following, we elaborate on the three technical dimensions with examples based on the ten queries above.

Data source dimension: It measures the type of input data the query is targeting. We have three types of input data in BigBench: structured, semi-structured and un-structured. For example, Query 1 uses semi-structured

Table 3: Business Categories Query Breakdown

Business category	Total	Percentage(%)
Marketing	18	60.0
Merchandising	5	16.7
Operations	4	13.3
Supply chain	2	6.7
New business models	1	3.3

web click streams as data source, while Query 3 does sentiment words extraction on un-structured product reviews data. In addition to using single data source, data source combinations are covered in the queries as well. For example, user click analysis (semi-structured) before store purchasing (structured) will join the two largest data sources, as is the case in Query 4.

Processing type dimension: It measures the type of processing appropriate for the query. This dimension covers the two common paradigms of declarative and procedural languages. In other words, some of our queries can be answered by declarative languages, others by procedural languages and others by a mix of both. In the scope of our benchmark, examples of declarative languages are SQL and similar constructs like Hive-QL. Map-Reduce is an example of a procedural language and Pig Latin has a mix of declarative and procedural constructs. Note that while some of the queries can be expressed in either declarative or procedural languages, there are queries that can only be expressed through procedural programming. In the former case, if the query is written through complex SQL constructs (e.g., window functions or user defined functions) we consider it a procedural query. However, queries that involve text analysis or sentiment analysis, like Query 3 and 7 fit in the later case as they have to be written using procedural programming. In the 10 queries above, Query 5, 6 and 9 can be written using SQL and thus are in the declarative category, while the other seven queries need procedural programming or a mix of procedural and declarative constructs.

Analytic technique dimension: It measures different techniques for answering business analytics questions. In general, we identified three major categories of analytic techniques: statistical analysis, data mining and simple reporting. Statistical analysis involves correlation analysis, time series, regression, etc. Statistical analysis is exemplified in Query 5, 9 and 10. For the data mining categories we use classification, clustering, association mining, pattern analysis and text analysis in our BigBench workload. Examples of data mining queries include Query 1, 2, 3, 4, 7 and 8. The reporting category is included in the BigBench as we believe that these queries represents a small but significant part of business analytics.This category covers the ad hoc queries and those that do not belong to statistical analysis or data mining. Most reporting queries are simple tasks that can be expressed in simple SQL. Note that most of our queries in the reporting category come from TPC-DS. Query 6 is an example of a reporting query.

While the query definition was driven by the business case represented by BigBench, their distribution over the three technical dimensions is believed to be reasonable and representative of the workload portrayed by the benchmark. We conclude this section by summarizing in Table 4 the query distribution along the three technical dimensions.

Table 4: Technical Dimensions Breakdown

Query processing type	Total	Percentage(%)
Declarative	10	33.3
Procedural	7	23.3
Mix of Declarative and Procedural	13	43.3
Data sources	**Total**	**Percentage(%)**
Structured	18	60.0
Semi-structured	7	23.3
Un-structured	5	16.7
Analytic techniques	**Total**	**Percentage(%)**
Statistics analysis	6	20.0
Data mining	17	56.7
Reporting	8	26.7

3.4 Metrics

Previous TPC benchmarks like TPC-H and recently TPC-DS have metrics based mostly on individual query execution times. The metric for BigBench could simply be the same or similar to either TPC-H or TPC-DS since from a high level it has similar phases, such as initial load, data refresh and query execution.

We defer the final design for the BigBench metric to future work. However, we believe that data loading and the type of processing dimension described in Section 3.3 is a necessary factor in BigBench's metric. Our rationale is that, on the one hand, DBMS and MR engines have different strengths in terms of loading, declarative and procedural processing. For example, Hadoop related systems are very efficient at loading and are generally optimized for MR processing. On the other hand, DBMS engines are optimized to process SQL, but MR/UDF processing and data loading may be less optimized. In addition, there is a recent effort for DBMS engines to process MR more efficiently, either natively or through an efficient co-existence with an MR engine (e.g., Hadoop, HIVE or Pig). One option to reflect the importance of the processing type dimension is to use the different processing types in the metric computation instead of using individual queries. Let T_L be the loading time, T_D the total time for queries in declarative processing, T_P the time for procedural processing queries and T_B the time for the remaining queries that have both declarative and procedural. A meaningful way of combining these four values in a composite metric is by computing their geometric mean as $\sqrt[4]{T\_L * T\_D * T\_P * T\_B}$. If the workload queries are used, the geometric mean could be calculated as $\sqrt[30]{\prod_{i=1}^{30} P_i}$ (where P_i denotes the execution time for $Query_i$).

4. EVALUATION

BigBench is targeted at DBMS and MR systems that claim to provide big data solutions. Therefore, any of those systems can be used to establish the feasibility of this benchmark. Standard DBMSes most likely will capture all data as relational tables by parsing the semi-structured data and establishing a schema. The un-structured data can also be captured as a table where the review text can be stored as VARCHAR or a blob column. Such DBMSes can implement our queries using SQL and some procedural constructs like UDF or even built in MR processing within the DBMS.

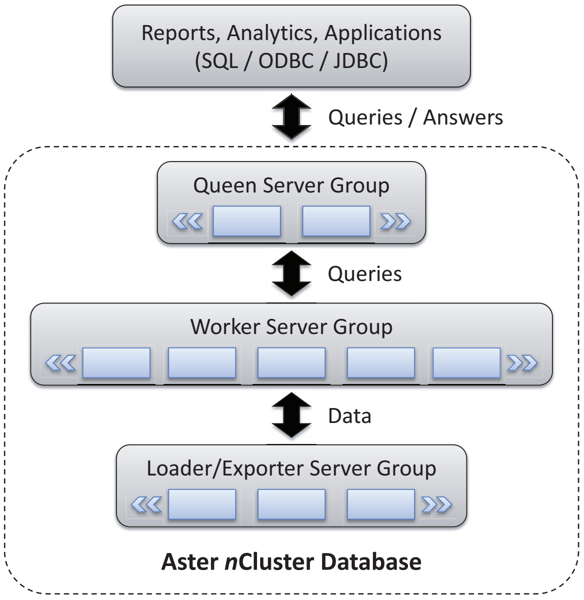

Figure 5: nCluster Architecture

Hadoop and its ecosystem with HIVE and Pig can also run BigBench. The data can be captured in HDFS or similar structures. The main strength of these systems is MR but they also have some relational operators like those in H-QL or Pig [2, 24]. Such relational operators can do joins, grouping and aggregations. BigBench can also be run on systems that have both DBMS and MR engines like Hadoop or any of its ecosystem products. Such systems consists most likely of a DBMS that connects or co-exists with an MR engine.

We chose to initially run BigBench on the Teradata Aster DBMS. TAD has all features needed to store and process big data. Data can be stored as tables and queries can be executed using the SQL-MR interface that extends declarative SQL with MR processing.

4.1 Teradata Aster DBMS

TAD is based on the nCluster technology. nCluster is a shared-nothing parallel database, optimized for data warehousing and analytic workloads [19]. nCluster manages a cluster of commodity server nodes, and is designed to scale out to hundreds of nodes and scale up to petabytes of active data.

Figure 4.1 depicts the nCluster architecture. Query processing is managed by one or more *Queen* nodes. These nodes analyze client requests and distribute partial processing among the *Worker* nodes. Each relation in nCluster is hash-partitioned (fact tables) or duplicated (dimension tables) across the Worker nodes to enable intra-query parallelism. Loading is done by special Worker nodes shown at the bottom of Figure 4.1.

In addition to database query processing, automated manageability functionality in nCluster allows adding new machines and redistributing data. The system performs automatic fail-over, retry of queries, and restoration of replication levels after a node failure. These features are essential in a large cluster of machines, where failures of various kinds occur regularly.

The SQL-MR supports a mix of SQL and polymorphic UDFs that process MR logic. The MR functions are parallelizable, self-describing and dynamically polymorphic where the function input schemas are determined implicitly at query execution time. Output schemas are determined programmatically by the function itself at query execution time as well. They are also equivalent to subqueries, making them subject to query optimization along with the other relations in a query. nCluster allows MR UDFs to be written using Java, C/C++, and scripting languages like Python.

4.2 End-to-End Execution

The test was executed on a 8 node Teradata Aster appliance. Each node is a Dell server with two quad-core Xeon 5500 at 3.07Ghz and hardware RAID 1 with 8 2.5" drives.

Due to time limitation, DSDGEN is used to produce the original TPC-DS tables in the structured part of our model. We used PDGF to generate the new parts of the data and the XML configuration for the review generator. The new parts produced by PDGF include the new Item_marketprice table, an apache-style web server log, and the online reviews. PDGF is also configured to match the references (PK-FK relationships) in the new data with the TPC-DS data. In the future, we plan on extending PDGF to handle the whole data generation aspects without the need for DSDGEN.

The data was loaded into TAD as tables. The web logs were parsed and converted to a table similar to the structure shown in Section 3.1. Product reviews are also interpreted as a table assuming the review text as a VARCHAR(5000).

As a proof of concept, we executed the workload as a single stream without velocity. Since we adapt the velocity methodology from TPC-DS adding it will not be difficult and can be implemented with a simple driver that adds data to the system periodically and re-submits a new stream of queries. Concurrent streams can also be handled similar to previous benchmarks like TPC-H.

The queries are written using TAD SQL-MR interface based on the description in Section 3.3. The reporting queries were written using SQL only and the rest were done through either an MR call or a mix of both SQL and MR. Below, we show the SQL-MR version of a sample of the 30 queries. The full list of the 30 queries written in SQL-MR can be found on our technical report that will be published with this paper. Note that all TAD MR functions used in the evaluation are part of a library TAD provides and packaged with the nCluster DBMS.

The query in Listing 4 is the SQL-MR equivalent of Query 3 in Section 3.3 which extracts sentiments and their polarity. The query retrieves from a reducer function called ExtractSentiment that takes input the source table product reviews. The call to the functions also specifies the column that has the text, the model for the sentiment analysis and the level of the search (sentence or word). The WHERE clause at the end picks positive or negative polarity.

The second example is for Query 1 as described in Section 3.3. The query is shown in Listing 5. It is the SQL-MR version equivalent for Query 1. It consists of 3 blocks. The most inner block is a SQL fragment that joins the web_sales and item tables and projects out category_id and customer_id. The inner block is fed as input to an MR function called basket_generator which finds the categories of pairwise items purchased together by customers. The input is partitioned by customer_id as specified by the PARTITION clause. The call to market_basket also specifies

```
SELECT pr_item_sk, out_content, out_polarity,
          out_sentiment_words
FROM ExtractSentiment
(
    ON product_reviews
    TEXT_COLUMN ('pr_review_content')
    MODEL ('dictionary')
    LEVEL ('sentence')
    ACCUMLATE ('pr_item_sk')
)
WHERE out_polarity = 'NEG' or out_polarity = 'POS';
```

Listing 4: Query 3

which field should the basket analysis be done on using the BASKET_ITEM clause. The last clause for the call to basket_generator is ITEM_SET_MAX(500) which limits the analysis to 500 pairs of items for each customer. The output of basket_generator is the input to the main query which basically finds the degree of affinity for each pair of categories.

```
SELECT
 category_cd1 AS category1_cd,
 category_cd2 AS category2_cd,
 COUNT(*) AS cnt
FROM
 basket_generator(ON
         (SELECT i.i_category_id AS category_cd,
                s.ws_bill_customer_sk AS customer_id
          FROM web_sales s
                INNER JOIN item i
                ON s.ws_item_sk = i_item_sk
         WHERE i.i_category_id is not NULL)
         PARTITION BY customer_id
                BASKET_ITEM('category_cd')
                ITEM_SET_MAX(500)
                )
GROUP BY 1,2
order by 1,3,2;
```

Listing 5: Query 1

The last example of our evaluation queries is Query 6 described in Section 3.3. The query is SQL only and adapted from the TPC-DS benchmark, it can be seen in Listing 6. As described before, Query 6 finds the impact of pricing change done on March 16, 1998. The query joins the following tables: web_sales used to capture sales done online, web_returns for returns of web_sales, warehouse which captures information about warehouses, item table that captures the products sold and date_dim which is a date lookup table. The join with web_returns is done as an outer join since not all orders have returns. The query computes the total sales before and after March 16, 1998 aliased as sales_before and sales_after in Listing 6. The query group on state location of the warehouse and ID of the items.

The run time of each of the 30 queries can be found at our technical report that will be published with this paper. Figure 7 lists the run time of the 10 queries used in the workload section. We also show the values of T_L, T_D, T_P and T_B as discussed in Figure 6. Note that we did not try any hardware or software optimizations to run the above 30 queries since our goal is to just make sure these queries run and produce meaningful results. The run time of the 30 queries varies from seconds to a little bit over an hour. This illustrates that we do not have a runway query situation and we also have a range of query complexities.

```
SELECT
   w_state,i_item_id
  ,sum(case when (cast(d_date as date) <
                  cast ('1998-03-16' as date))
          then ws_sales_price -
               coalesce(wr_refunded_cash,0)
             else 0 end)
    as sales_before
  ,sum(case when (cast(d_date as date) >=
                  cast ('1998-03-16' as date))
          then ws_sales_price -
               coalesce(wr_refunded_cash,0)
             else 0 end) as sales_after
 FROM
   web_sales left outer join web_returns on
       (ws_order_number = wr_order_number
        and ws_item_sk = wr_item_sk)
  ,warehouse, item, date_dim
 WHERE
    i_item_sk          = ws_item_sk
    and ws_warehouse_sk    = w_warehouse_sk
    and ws_sold_date_sk    = d_date_sk
    and d_date between
      (cast ('1998-03-16' as date) - interval '30
           day')
        and (cast ('1998-03-16' as date) + interval
             '30 day')
 GROUP by w_state,i_item_id
 ORDER by w_state,i_item_id;
```

Listing 6: Query 6

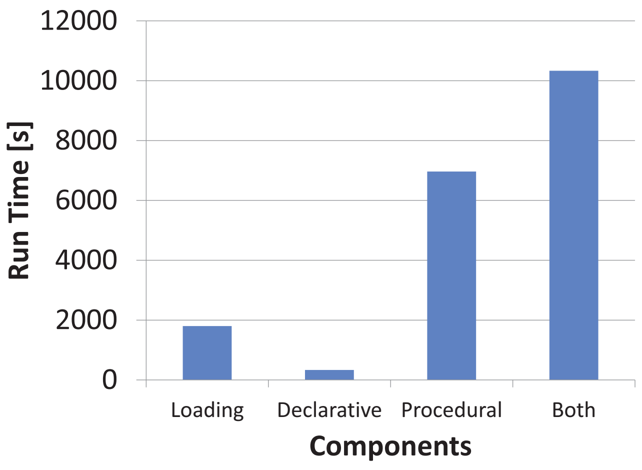

Figure 6: Runtime of Metric Components

5. CONCLUSION

In this paper we presented BigBench, a proposal for an end-to-end big data benchmark. The proposal covers a data model addressing the velocity, variety and volume common in big data. Velocity is accomplished by continuous feed into the data store while variety is addressed by including structured, semi-structured and unstructured in the data model. The data model also can scale to large volumes based on as scale factor. We used PDGF as a starting point for our data generator that covers the structured part. PDGF is enhanced to produce the semi-structured and unstructured data. The unstructured component is based on a novel technique we developed leveraging the Markov chain model. The proposal also provides a comprehensive list of workload queries and sets directions for a novel metric that focuses on the different types of processing in big data. Finally, we

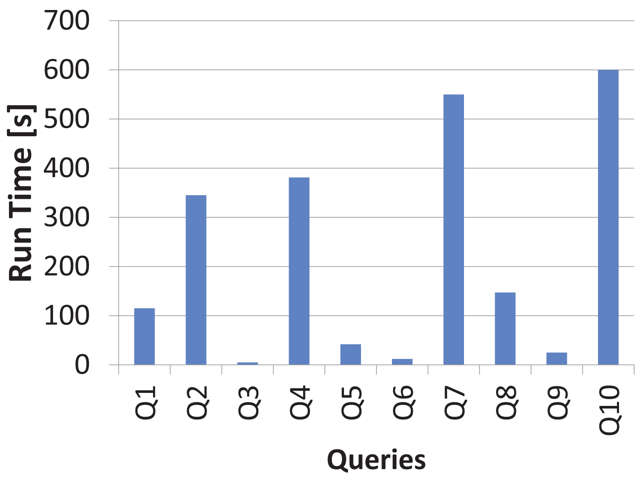

Figure 7: Runtime of Sample Queries

verified the feasibility and applicability of our proposal by implementing and running it on Teradata Aster DBMS.

For future work, we are planning to extend this work in three main areas. First, we would like to enhance the proposal to be a concrete specification that can lead to an industry standard benchmark. This work include finalizing and detailing the data, workload and metric specifications. We also think system availability during failure should be addressed in the final specification. Second, we think it will be useful to provide a downloadable kit that can be used to setup and run the benchmark. This work include finalizing the implementation of our data and query generators. Finally, we are planning to extend the benchmark proof of concept to include velocity and multi-user test. We also would like to run the benchmark on one the Hadoop ecosystem like HIVE.

6. REFERENCES

[1] Apache Hadoop Project. `http://hadoop.apache.org`.

[2] Apache Hive Project. `http://hadoop.apache.org/hive`.

[3] Cloudera Distribution Including Apache Hadoop (CDH). `http://www.cloudera.com`.

[4] Greenplum Database. `http://www.greenplum.com`.

[5] GridMix Benchmark. `http://hadoop.apache.org/docs/mapreduce/current/gridmix.html`.

[6] Oracle Database - Oracle. `http://www.oracle.com`.

[7] PigMix Benchmark. `https://cwiki.apache.org/confluence/display/PIG/PigMix`.

[8] Teradata Database - Teradata Inc. `http://www.teradata.com`.

[9] TwinFin - Netezza, Inc. `http://www.netezza.com/`.

[10] TPC Benchmark DS, 2012.

[11] J. Bentley. *Programming Pearls*. Addison-Wesley, 2000.

[12] M. J. Carey, D. J. DeWitt, and J. F. Naughton. The oo7 Benchmark. In P. Buneman and S. Jajodia, editors, *SIGMOD'93*, pages 12–21. ACM Press, 1993.

[13] M. J. Carey, D. J. DeWitt, J. F. Naughton, M. Asgarian, P. Brown, J. Gehrke, and D. Shah. The BUCKY Object-Relational Benchmark (Experience Paper). In *SIGMOD*, pages 135–146, 1997.

[14] M. J. Carey, L. Ling, M. Nicola, and L. Shao. EXRT: Towards a Simple Benchmark for XML Readiness Testing. In *TPCTC*, pages 93–109, 2010.

[15] C. Chambers, A. Raniwala, F. Perry, S. Adams, R. R. Henry, R. Bradshaw, and N. Weizenbaum. FlumeJava: Easy, Efficient Data-Parallel Pipelines. In *PLDI*, pages 363–375, 2010.

[16] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking Cloud Serving Systems with YCSB. In *SoCC*, pages 143–154, 2010.

[17] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[18] M. Frank, M. Poess, and T. Rabl. Efficient Update Data Generation for DBMS Benchmark. In *ICPE*, 2012.

[19] E. Friedman, P. Pawlowski, and J. Cieslewicz. SQL/MapReduce: A Practical Approach to Self-Describing, Polymorphic, and Parallelizable User-Defined Functions. *PVLDB*, 2(2):1402–1413, 2009.

[20] J. Gray. GraySort Benchmark. Sort Benchmark Home Page – `http://sortbenchmark.org`.

[21] D. Laney. 3D Data Management: Controlling Data Volume, Velocity and Variety. Technical report, Meta Group, 2001.

[22] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. 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`.

[23] R. O. Nambiar and M. Poess. The Making of TPC-DS. In *VLDB*, pages 1049–1058, 2006.

[24] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A Not-So-Foreign Language for Data Processing. In *SIGMOD*, 2008.

[25] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. Lopez, G. Gibson, A. Fuchs, and B. Rinaldi. YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In *SoCC*, pages 9:1–9:14, 2011.

[26] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis. In *SIGMOD*, pages 165–178, 2009.

[27] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the Data: Parallel Analysis with Sawzall. *Scientific Programming*, 13(4):277–298, 2005.

[28] M. Pöss, R. O. Nambiar, and D. Walrath. Why You Should Run TPC-DS: A Workload Analysis. In *VLDB*, pages 1138–1149, 2007.

[29] T. Rabl, M. Frank, H. M. Sergieh, and H. Kosch. A Data Generator for Cloud-Scale Benchmarking. In *TPCTC*, pages 41–56, 2010.

[30] T. Rabl, M. Sadoghi, H.-A. Jacobsen, S. Gómez-Villamor, V. Muntés-Mulero, and S. Mankowskii. Solving Big Data Challenges for Enterprise Application Performance Management. *PVLDB*, 5(12):1724–1735, 2012.

[31] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *VLDB*, pages 974–985, 2002.

[32] Teradata Aster. *Teradata Aster Big Analytics Appliance 3H - Analytics Foundation User Guide*, release 5.0.1 edition, 2012. `http://www.info.teradata.com/edownload.cfm?itemid=123060004`.

[33] L. Wyatt, B. Caufield, and D. Pol. Principles for an ETL Benchmark. In *TPCTC*, pages 183–198, 2009.