Analysis of TPC-DS - the First Standard Benchmark for SQL-Based Big Data Systems

Meikel Poess
Server Technologies - Oracle Corporation
Redwood Shores, California, USA
meikel.poess@oracle.com

Tilmann Rabl
DIMA Group - TU Berlin
Munich, Germany
rabl@tu-berlin.de

Hans-Arno Jacobsen
Application and Middleware Systems Group - TU Munich
TU Munich, Germany
jacobsen@in.tum.de

ABSTRACT

The advent of Web 2.0 companies, such as Facebook, Google, and Amazon with their insatiable appetite for vast amounts of structured, semi-structured, and unstructured data, triggered the development of Hadoop and related tools, e.g., YARN, MapReduce, and Pig, as well as NoSQL databases. These tools form an open source software stack to support the processing of large and diverse data sets on clustered systems to perform decision support tasks. Recently, SQL is resurrecting in many of these solutions, e.g., Hive, Stinger, Impala, Shark, and Presto. At the same time, RDBMS vendors are adding Hadoop support into their SQL engines, e.g., IBM's Big SQL, Actian's Vortex, Oracle's Big Data SQL, and SAP's HANA. Because there was no industry standard benchmark that could measure the performance of SQL-based big data solutions, marketing claims were mostly based on "cherry picked" subsets of the TPC-DS benchmark to suit individual companies strengths, while blending out their weaknesses. In this paper, we present and analyze our work on modifying TPC-DS to fill the void for an industry standard benchmark that is able to measure the performance of SQL-based big data solutions. The new benchmark was ratified by the TPC in early 2016. To show the significance of the new benchmark, we analyze performance data obtained on four different systems running big data, traditional RDBMS, and columnar in-memory architectures.

CCS CONCEPTS

• **Information systems → Database performance evaluation**;

KEYWORDS

TPC-DS, benchmark, big data

ACM Reference Format:
Meikel Poess, Tilmann Rabl, and Hans-Arno Jacobsen. 2017. Analysis of TPC-DS - the First Standard Benchmark for SQL-Based Big Data Systems. In *Proceedings of ACM Symposium of Cloud Computing conference, Santa Clara, California USA, September 25-27 (SoCC'17),* 13 pages.
https://doi.org/10.475/123_4

SoCC'17, September 25-27, Santa Clara, California USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

1 INTRODUCTION

The big data revolution, triggered by the availability of powerful yet affordable commodity hardware running open source software stacks, is starting to provide a viable alternative to traditional RDBMS technologies. Initially, only large Web 2.0 companies, such as Facebook, Google, and Amazon deployed systems based on Hadoop and related tools, such as YARN [31], MapReduce [9], Pig [23], and NoSQL databases. The amount of coding necessary to perform decision support tasks on these systems, such as data mining, predictive analytics, text analytics and statistical analysis, prevented a wide commercial adaptation of these technologies.

Only when higher level languages were added on top of Hadoop and MapReduce the wide adaptation of these technologies in decision support installations was triggered. Many big data solutions are moving away from the pure NoSQL model to a not-only-SQL approach resulting in an explosion of SQL-based implementations, which are designed to support big data on the Hadoop ecosystem, e.g., Hive [27], Stinger [14], Impala [6], Shark [32], Presto [11], and Spark [17]. Many RDBMS vendors are following suit by adding Hadoop support into their SQL engines, e.g., IBM's Big SQL [1], Oracle's Big Data SQL [2], and SAP's Vora [3].

Many reasons drive the use of SQL in big data solutions. It is intuitive to write data analysis queries in SQL, because most users think about data as being organized in two dimensional tables, i.e., spreadsheets. The declarative nature of SQL increases developer productivity and code maintainability, because writing a query in SQL requires the description of a result set rather than the algorithm to compute it.

In need for benchmarks to showcase the performance of their big data SQL engines many vendors used custom benchmarks, which they derived from the first version of the Transaction Processing Performance Council's (TPC) benchmark TPC-DS (V1). Rather than formally running an entire TPC-DS V1 according to its specification and publishing fully certified results, vendors cherry-picked those portions of the TPC-DS V1 benchmark that made their particular brand of technology excel, ignoring the general use case TPC-DS V1 was designed to test. Many marketing publications used a subset of the schema and queries, executed the benchmark in a special way, and reported a metric that positions a system in the best possible light [8, 10, 13, 19]. This kind of "benchmarketing" is not new to the industry and it is precisely what triggered the founding of the TPC 25 years ago.

Instead of questioning the credibility of these highly customized claims and fining vendors for violating its fair use polices, we redesigned the existing TPC-DS V1 to create a fair and comprehensive benchmark that specifically targets the performance measurement

of big data SQL systems. The TPC branded the new benchmark TPC-DS V2 to enhance market recognition. TPC-DS V2 specifically addresses the domain of SQL-based big data systems.

The major contributions of this paper are: (i) to present TPC-DS V2, a new TPC benchmark for measuring the performance of SQL based big data solutions, (ii) to provide a deep analysis of V2 with emphasis on its key features, design decisions and workload differences to V1, and (iii) to analyze the run time behavior (elapsed times and resources consumption) of all 99 queries during single- and multi-user runs on four different setups that resemble the diversity of systems being deployed in big data solutions.

The remainder of this paper is organized as follows. The following section presents related benchmarks. Section 3 touches on the key features of TPC-DS V2, necessary to understand the remainder of the paper, to motivate the major changes in Version 2, that allow it to measure the performance of SQL based big data solutions and to analyze its major challenges. Section 4 provides experimental results of four solutions that are deployed in the big data use case TPC-DS V2 aims to address, including a detailed analysis of the experimental results. The paper concludes with Section 5.

2 RELATED WORK

TPC-DS V2 is the first industry standard benchmark for measuring the performance of SQL-based big data systems. It is based on TPC-DS V1, which has been widely used in academia and industry to analyze analytic features of SQL-based database engines. While there have been two publications describing TPC-DS V1 [21, 25], they describe early versions of the benchmark specifications and do not contain modifications made to the query set, metric, executions rules, and the update model, before TPC-DS V1 was ratified as an industry standard benchmark. This paper describes the second version of the TPC benchmark in its released form and touches on the difference to Version 1.

Several other benchmarks have been proposed for big data benchmarking. In the following, we discuss only benchmarks directly related to TPC-DS V2. We do not discuss other types of big data benchmarks, e.g., domain specific benchmarks (graph-based benchmarks [5, 20]) or benchmarks targeted only at a single type of systems (MapReduce [16] or Spark [17]).

TPC-DS V2 is not the first big data benchmark released by the TPC. Having identified the need for big data benchmarking, the TPC released TPCx-HS, a benchmark based on the Hadoop/MapReduce Terasort implementation, as a stopgap solution [22]. TPCx-HS was the TPC's first kit-based benchmark and as such has a fully implemented kit that can be easily run on publicly available Hadoop/MapReduce distributions. Unlike most TPC benchmarks, TPCx-HS is not an application level benchmark that simulates a use case realistically, but a component benchmark that targets mainly the storage system and sorting engine.

Another big data benchmark, recently released by the TPC, is TPCx-BB, which is based on BigBench [12, 26], which in turn is partially based on TPC-DS V1. Instead of stressing the SQL capabilities of an engine, it allows for non-SQL implementations and engine specific optimizations. BigBench has only 30 queries, 10 of which are based on TPC-DS V1 queries. The remaining queries address use cases that are difficult to be expressed in standard SQL. BigBench targets read-only big data platforms in general with some SQL aspects, while TPC-DS V1 is specifically designed for an in-depth read/write performance testing of SQL engines.

Most other benchmarks for big data systems are suites of small workloads which are much less complex than the workloads of TPC-DS and TPCx-BB. Popular examples are Pavlo's benchmark [24] and its successors HiBench [15] and the AMPLab Benchmark [4]. These benchmarks consist of dependent workloads with only small SQL parts, comprised of simple filter and aggregation queries. Today, TPC-DS V2 is the most comprehensive and most complex benchmark, not only for SQL-based big data systems but any kind of SQL engines as well.

3 BENCHMARK ANALYSIS

This section presents and analyzes the key features in the new version of TPC-DS (V2), while highlighting the differences to the old version (see [29] for the complete specification).

3.1 Paradigm Shift in Data Ownership

Big data systems, many of which are based on the Hadoop ecosystem, introduced a paradigm shift in data ownership. Traditionally, only one system had control over a given data set, namely the DBMS. This control enables the use techniques and algorithms to implement performance enhancements and to enforce data correctness that rely on persistent auxiliary data structures. For instance, the uniqueness of primary keys in a table is usually guaranteed by enforcing a constraint, which in turn uses a primary key index for efficiency.

Big data systems follow an open data approach, in which all products in its ecosystem, including MapReduce, can access and modify the same full-fidelity data sets, mostly saved in HDFS. While this approach eliminates the costly process of copying and converting data into different formats, it makes concepts like enforcement of constraints impractical because the query engine does not necessarily know immediately when the table data is modified by another product. TPC-DS V2 allows constraints to be non-enforced, because query optimizers of most SQL engines rely on understanding basic data characteristics, such as primary-foreign key relationships and not-null constraints, so that they generate reasonable query plans. Being able to operate on raw data also blurs the definition of what constitutes a database load (see Section 3.5).

3.2 Goodbye ACID - Welcome BASE

TPC-DS V1 requires full ACID compliance. It must be demonstrated before any benchmark result can be published by running functional tests on a similar, but much scaled-down database. Due to de-coupling of the ownership of data from the processing of data, big data solutions are inherently not ACID, but BASE compliant (Basically Available, Soft state, Eventual consistency), i.e., they guarantee some level of data accessibility through data mirroring.

Instead of ACID, TPC-DS V2 requires a more relaxed version of durability, which it refers to as *data accessibility*. To satisfy *data accessibility* a system must continue executing queries and data integration functions with full data access during and after a permanent irrecoverable failure of any single durable medium containing any database objects, e.g., tables, explicit auxiliary data structures, or metadata. With large cluster configurations being common in big

data installations, including a node failure test seems obvious. However, since the benchmark does not require multi-node configurations and ACID is required to recover transactions from complete system failures, a node failure is not included in TPC-DS V2.

3.3 Periodic Data Integration Workload

A DS refresh process usually involves data extraction, data transformation, and data load commonly referred to as data integration (DI). The data extraction step extracts pertinent data from production OLTP databases or other relevant data sources. The transformation step cleans the extracted data. The data load step performs the actual insertion, modification, and deletion of decision support database table data.

TPC-DS V1's required a full implementation of a DI process for all non-static tables, i.e., refresh of history and non-history keeping dimension, inserts into fact tables, and deletes from fact table. Realizing that typical big data implementations of DSS do not necessarily undergo such a rigid data integration process, the TPC decided to focus on the adding and removing of fact table data in TPC-DS V2. Not requiring the maintenance of dimension tables eliminates the need for update statements, which would have forced big data systems without native support for update statements to implement them as *deletes and inserts*. Due to its slow execution, implementing updates as deletes and inserts would have put big data systems at a significant disadvantage and, consequently, discouraged rapid adoption of the benchmark by new technologies.

The remaining insert and delete operations on fact tables are believed to be sensible and adequate enough for analyzing the performance of data integration operations currently deployed in big data implementations of DSS. The insert and delete operations logically delete old facts, e.g., old sales transactions to make room for new fact data, i.e., new sales transactions. The intention of these operations is to exercise both range and scattered deletes. The deletion of fact table data can be implemented by dropping database objects, (files in case of Hadoop-based systems), if the corresponding data is clustered based on date ranges. Inserts then simply recreate the deleted database objects with new content. On the other hand, the delete operations on returns fact table are always scattered since returns can occur in a three month window after their corresponding sales and clustering.

3.4 Query Workload

TPC-DS utilizes a generalized query model that addresses the variety of queries found in today's big data systems. The queries cover the interactive and iterative nature of on-line analytical processing (OLAP), long-running, complex data mining tasks, knowledge discovery, and frequent reports. Amalgamating these different query types, especially ad-hoc and reporting into one benchmark is achieved by allowing ddl-driven performance enhancement techniques, such as partitioning or materialized views, only on a subset of the data. Queries referencing tables with performance enhancing techniques are then classified as reporting queries, others are ad-hoc queries.

While most queries are carried over from TPC-DS V1 to TPC-DS V2, some were modified. Most of the modification address inconsistencies between the functional query definition (SQL text) and the business description in the specification, non-deterministic query results, bugs and equivalent rewrites to allow more big data products to run the queries. Table 1 lists the major query changes.

Table 1: Query modifications applied to V2

Queries	Modification
10, 35	Rewrote disjunctions of exist predicates into exist predicates of unions, e.g. `exists (SubQuery1) OR exists (SubQuery2)` into `exists (SubQuery1 UNION ALL SubQuery2)`
34, 56, 64, 73, 75, 76	Added additional columns in order by to make query output deterministic
59	Corrected wrong ratio `tue_sales1/tue_sales1` into `tue_sales1/tue_sales2`
77	Added `group by cr_call_center_sk` to `catalog_returns` common subexpression
78	Refered to coalesce expression in `ORDER BY` by name rather than repeating the expression, change `coalesce(ws_qty,0)>0 and coalesce(cs_qty, 0)>0` to `(coalesce(ws_qty,0)>0 or coalesce(cs_qty, 0)>0)` and `(coalesce(ws_qty+cs_qty,1)` to `(coalesce(ws_qty,0)+coalesce(cs_qty,0))` and correced wrong join clause `left join cs on (cs_sold_year=ss_sold_year and cs_item_sk=cs_item_sk and cs_customer_sk=ss_customer_sk)` to `left join cs on (cs_sold_year=ss_sold_year and cs_item_sk=ss_item_sk and cs_customer_sk=ss_customer_sk)`
84	Added explicit coalesce around columns in concatenation expressions due to some query engines evaluating columns concatenations to $NULL$ if one part of the expression is $NULL$

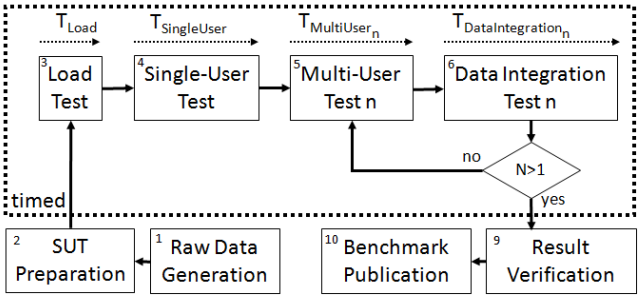


Figure 1: High Level Execution Rules

3.5 Metric and Execution Rules

The execution rules and the metric of a benchmark specification are intrinsically connected to each other and they are equally powerful in how they emphasize performance aspects of a system and, ultimately, drive performance innovation. To establish an unambiguous ranking of result, each TPC benchmark specification is required to contain only one primary performance metric. The execution rules and metric in V2 have been redesigned to emphasize the performance characteristics of big data systems.

Figure 1 illustrates the timed and the un-timed phases of the execution of a TPC-DS V2 benchmark run. The generation of the raw data

Table 2: Differences between TPC-DS V1 and V2

	TPC-DS V1	TPC-DS V2
Data ownership	Inside DBMS	Outside DBMS
Database load	Conversion into proprietory format	Simple text copy
Transactional properties	ACID	BASE
Node failure required	Yes	No
Number of queries	99	99, 12 modified
Updates	Most tables	Fact tables only
Trickle updates	Optional	Disallowed
Cloud pricing	No	Yes
Performance metric	Arithmetic mean	Geometric mean

set, i.e., flat files (Step 1) and the preparation of the System Under Test (SUT) in Step 2 are not timed. The size of the raw data is determined by the scale factor $SF \in 100, 300, 1000, 3000, 10000, 30000, 100000$, which represents the data size in GB. The timed portion starts with the execution of the load test (Step 3), followed by the single-user test, aka *power test*, (Step 4) and two pairs of multi-user tests, aka *throughput tests*, (Step 5) and data integration test (Step 6). Steps 5 and 6 are executed twice to measure the impact of updates to the system after the first data integration test.

The load test (T_{Load}) entails all steps necessary to prepare the SUT to execute the subsequent performance tests. Due to TPC-DS V2 being technology agnostic, the individual steps of the load test are not explicitly listed. However, validation of unenforced constraints, if defined, and all requirements to assure BASE properties, including synchronizing loaded data on RAID devices and taking database backups, if necessary, are part of the load test. Additionally, the open data paradigm (see Section 3.1) questions whether a load time is necessary at all. One could argue that systems that efficiently query ext data, i.e., without performing costly conversions into secondary formats (Parquet, AVRO, JSON, etc.) have a load time of zero. However, because the load time is part of the geometric mean metric, it could cause the metric to increase dis-proportionally if the load time approaches zero, essentially breaking the metric. TPC-DS V2 counters this problem by defining the notion of a *database location*, e.g., HDFS, and including the time it takes to place text data into the database location into the load time. Consequently, at a minimum, the load test performs the placing of the TPC-DS V2 flat files into the database location, which, on many big data platforms, translates into copying data to HDFS.

The single-user test ($T_{SingleUser}$) executes all 99 queries consecutively in a single session, measuring a system's ability to maximize system utilization and minimize query response time. The multi-user test ($T_{MultiUser_n}$, with $n \in 1, 2$) is followed by the data integration test. Both are executed twice. Each multi-user test executes s concurrent sessions, which in turn each executes all 99 queries consecutively (s must be an even number). Each session executes queries in a different permutation to prevent the use of unrealistic data caching. The multi-user tests measure a system's ability to divide resources among concurrent sessions to maximize overall query throughput.

The data integration tests consist of inserting new sales/returns data and deleting old sales/returns data. The data integration test measures the systemâĂŹs ability to periodically ingest new data and purge old data. It is run immediately after each multi-user run to reveal any query performance implications in the following multi-user run that may occur due to the maintenance of auxiliary data structures, a different data layout or changes in statistics. The elapsed time of the data integration tests are denoted as $T_{DataIntegration_n}$, with $n \in 1, 2$.

Data and *query result* caching techniques are not explicitly prohibited in TPC-DS V2, because their use is in general extremely difficult to police and to some extend desirable as a performance differentiator. However, the queries and execution rules are designed to make such caching unprofitable beyond its typical use in real world systems. Selectivity predicates for each query $Q_{i,j}$ generated from template T_i are generated at random to cover the entire range of possible values, thereby limiting the use of data and query result caching.

The performance metric in Version 2, *QphDS@SF*, has been changed from an arithmetic mean to a geometric mean of the four elapsed times of the above tests, despite the pros and cons of using geometric means to calculate a single number to represent performance [7]. This change was done to address concerns by some TPC member companies that the original metric could, for some implementations, be dominated by data maintenance and load.

$$QphDS@SF = \left\lfloor \frac{SF * Q}{\sqrt[4]{T_{PT} * T_{TT} * T_{DI} * T_L}} \right\rfloor \quad (1)$$

The nominator is the SF multiplied by the total number of queries, executed by all S_q concurrent users, $Q = S_q * 99$. The denominator is the geometric mean of the elapsed times of all performance tests: $T_{PT} = T_{SingleUser} * S_q$, $T_{TT} = T_{MultiUser_1} + T_{MultiUser_2}$, $T_{DI} = T_{DataMaintenance_1} + T_{DataMaintenance_2}$, and $T_L = 0.01 * S_q * T_{Load}$. The elapsed times for the load and single-user tests are multiplied by the number of concurrent users, S_q, to avoid for the multi-user component to become the dominant contributor to the metric for large S_q.

4 EXPERIMENTAL RESULTS

We run our tests against four different setups, which resemble the diversity of systems being used to run DSS workloads. Because of licensing restrictions of the commercial systems we are evaluating in this study, we cannot disclose absolute numbers. Hence the results are anonymized: *Setup A* and *Setup B* are big data systems with their own storage engine on top of HDFS. Both setups use identical hardware, namely nine nodes, each with 96GB of RAM, 12 high capacity SAS drives, 2 sockets with 8 cores and 16 threads. The total configuration has 864GB of RAM, 108 SAS drives and 288 threads. *Setup C* is a traditional RDBMS system using its own storage format. This setup uses 14 nodes, each with 256GB of RAM, 13 high performance SAS drives, 2 sockets with 6 cores and 12 threads each. The total configuration has 3.5TB of RAM, 168 high performance SAS drives and 336 threads. Setup D is a columnar organized in-memory solution. It runs on a single SMP system with 2TB of RAM, 24 high capacity SAS disks and 8 sockets, each with 18 cores and 36 threads (288 threads total). Using data compression this configuration is capable of keeping the entire 3000GB database

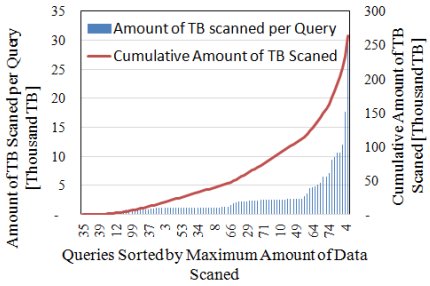


Figure 2: Upper Bound of Data

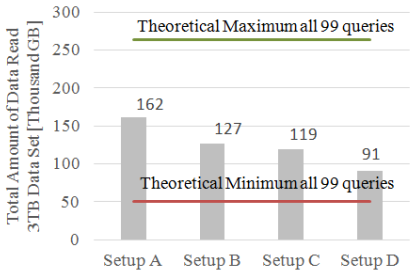


Figure 3: Minimum Maximum Data Access

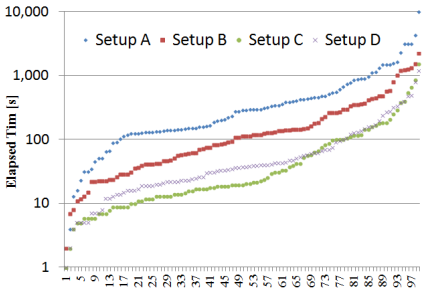


Figure 4: Sorted Elapsed Times

in RAM. All setups are able to perform intra-query as well as inter-query parallelism and provide mechanisms to spill intermediate result sets to disks that are too big to keep in memory.

We do not claim to be experts in tuning all of the above setups. To conduct a fair comparison among these setups, we perform our measurements "out of the box". We measure the performance immediately after installation without any configuration, modification, or creation of auxiliary data structures. The numbers presented in this section are the best out of three runs.

4.1 Data Scan Analysis

One of the main system factors of resource consumption in answering big data queries is the sifting through vast amounts of data in order to identify those parts that are required to answer them. Minimizing the amount of data scanned, avoiding reading data multiple times, and performing the remaining scans quickly are key differentiators among big data solutions. The following paragraphs establish lower and upper bounds for the amount of data the entire TPC-DS V2 query workload requires.

The amount of data that is needed to compute a given query result depends not just on the complexity of the query, but also on how optimally the generated query plan can be executed on a given system, whether the system supports horizontal table pruning to eliminate unnecessary rows, e.g., partitioning and indexing, and whether the system supports vertical pruning of tables to eliminate unnecessary columns, e.g., columnar access.

We establish an estimate for the upper bound of data needed to compute a query result by assuming all tables referenced by the query are accessed fully, i.e., no vertical or horizontal table pruning and disregarding common subexpression elimination of *with clauses*, which may result in multiple reference counts of the same table in a given query. We discount join methods that may result in multiple scans of the same table, e.g., nested loop join without index. Since traditional indexing is not common in big data solutions, this is a fair assumption. Denoting TQ_i as the subset of tables accessed by query i, $ARL(t)$ as the average row length for table t in bytes and $C(t)$ as the cardinality of table t, then the amount of data required by a given query is the sum of all table references multiplied by their sizes in bytes of raw data, i.e., uncompressed data as generated by the data generator, dsdgen. We estimate the upper bound of data that is required to answer all 99 queries as:

$$\sum_{i=1}^{99} \sum_{t \in TQ_i} ARL(t) * C(t) \approx 260,000GB \quad (2)$$

We establish an estimate for the lower bound of data needed to compute a query result by assuming tables are accessed after horizontal table pruning, only referenced columns are accessed in each table and each table's data is only accessed once. The minimum amount of data a query accesses for any table it references is the table cardinality after horizontal table pruning multiplied by the sum of the average raw column length in bytes of all columns referenced. Denoting C_t as the set of columns referenced in table t, $C_{hp}(t)$ as the cardinality of a table t after horizontal pruning and $ACL(c)$ as the average column length of column c, then we can estimate the lower bound for the amount of data read to answer all 99 queries as:

$$\sum_{i=1}^{99} \sum_{t \in TQ_i} \sum_{c \in C_t} ACL(c) * C_{hp}(t) \approx 50,000GB \quad (3)$$

Figure 2 summarizes the upper bound of data needed to answer each of the 99 queries at scale factor 3000 (bar chart with y-axis on the left). Please note that these should not be treated as actual bounds or min/max values, but as estimates. The range of data scanned in each query varies widely from 394MB to 30TB. 17 queries access less than 0.5TB each, 11 queries access between 0.5TB and 1TB each, 29 queries access between 1TB and 2TB each, 25 queries access between 2TB and 3TB each, 5 queries access between 3TB and 5TB each, 7 queries access between 5TB and 10TB each and 5 queries access up to 30TB each. The line chart (y-axis on the right) shows the cumulative data amount accessed by all queries.

Figure 3 shows the amount of data the four different systems read during the execution of all 99 queries and compares that to the theoretical Maximum established in earlier paragraphs. Setup A reads 162,987GB, Setup B reads 126,974GB, Setup C reads 118,808GB and System D reads 90,208 GB. Because none of the systems reads the theoretical maximum of about 260,000GB during query execution, it is to assume that each system performs some data pruning technique. However, none of the system reduces the amount of data read to the theoretical minimum of about 50,000GB. We should also note that there is a very large discrepancy between the system reading the most and that reading the least amount of data of about 70,000GB.

4.2 Single-User Test Analysis

The single-user test, a.k.a., the *Power Test*, executes queries consecutively in one session. Its purpose is to measure how well a particular system is able to minimize the aggregated query elapsed times. For a benchmark query set to be considered meaningful with respect to a single-user run, it not only needs to challenge all resources of a system, but it also needs to be diverse enough to be able to reveal a system's particularities. Queries that execute in similar elapsed times on systems need to be further investigated whether they provide additional value to the benchmark. First, we discuss the distribution of query elapsed times of the four setups.

In our implementation of the power test we assign all available resources of a setup to the execution of one query at any given time. Setup A finishes the power test in 39,763s, Setup B in 14,392s, Setup C in 6,222s and Setup D in 4,261s.

Figure 4 shows the sorted elapsed times of all four setups on a logarithmic scale. The elapsed times of Setup A vary between 21s and 3,178s. The elapsed times of Setup B vary between 7s and 2,236s and those of Setup C vary between 2s and 1,531s. While Setup D outperforms all other setups for most queries, two of its queries perform equally long as on the other systems.

Although the graphs of the four setups never cross, we cannot conclude that Setup D executes each query faster than the other setups, as the queries are sorted on elapsed time. However, from the graphs we can establish a total elapsed time ranking among the setups and conclude that the elapsed times vary dramatically on each setup, between just a few seconds to over one hour, indicating that the benchmark is doing a good job in making each system struggle to execute some queries.

Ultimately, the queries must be able to categorize systems in meaningful ways and to reduce the overhead of running the benchmark, the set should be minimalistic. To determine whether the queries are able to assess today's systems with respect to a single user test, we analyze the data we collected on each system using: (i) Coefficient of Variation Analysis, (ii) K-means Cluster Analysis, and (iii) Normalized Elapsed Time Analysis.

4.2.1 Intra Setup Coefficient of Variation Analysis. To measure how much the single-user elapsed times fluctuate, we calculate their coefficient of variation for each setup. Because the coefficient of variation is a unit-free measure of relative variability, we can use it to assess the variation of the elapsed times of one system, as well as the elapsed time variations across multiple systems.

The coefficient of variation of the query elapsed time distribution of Setup s is defined as the ratio of the standard deviation σ to the mean μ of all elapsed times (CV=$\frac{\sigma}{\mu}$). The coefficient of variation of the elapsed time distributions of Setup A is 1.36, of Setup B it is 1.64, of Setup C is 2.49, and of Setup D is 4.37, which means that query elapsed times on Setup D vary the most, namely, 4.37 times from the mean. Because all systems show a coefficient of variation larger than 1, their elapsed times distribution is considered statistically high-variance. This is a good indication that the query set in its entirety is diverse enough to challenge systems in different ways. As a comparison, the coefficient of variation of the top published TPC-H result for each scale factor (100GB, 300GB, 1000GB, 3000GB, 10000GB, 30000GB and 100000GB) vary between 0.69 and 1.2 (see [30]).

4.2.2 Intra Setup K-means Analysis. The coefficient of variation analysis gives us a good idea on how disparate the values in each elapsed time distribution are. We analyze the distributions further to see whether we can divide the elapsed times into homogeneous query classes. A division of the elapsed times into a small number of classes would indicate redundancy in the query set. We analyze the elapsed time distribution of the queries run during the power run (P) for each system $s \in \{A, B, C, D\}$. Using the k-means algorithm [18] we calculate the elapsed time centroids $C_{P,s,k}$ and clusters $Q_{P,s,k}$ for $k \in \{2, 4, 6, 8, 10\}$. To determine the best value for k we use the elbow method [28]. Our dataset size is small enough to find the elbow for each system by visually inspecting the graphs. For each value of k and setup s during the power run P we calculate the sum of squared errors ($SSE_{P,s,k}$). First we calculate the mean as:

$$\mu_{P,s,k} = \frac{1}{|Q_k|} \sum_{q \in Q_k} T_{P,s}(q) \tag{4}$$

We then calculate the SSEs as:

$$SSE_{P,s,k} = \sum_{k=1}^{k} \sum_{q \in Q_k} (\mu_{P,s,k} - T_{P,s}(q))^2 \tag{5}$$

Figure 5 graphs the SSEs for $k \in 2, 4, 6, 8, 10$ and each setup $s \in \{A, B, C, D\}$. The "elbow" for each setup is at $k = 4$. That means our k-means analysis divides the query elapsed times, and therefore the queries for each setup into four categories: (i) fast-, (ii) medium-, (iii) , slow- and (iv) very-slow-running queries. This seem to be a very coarse granularity, but we can have a look at how many queries are in each category and whether we can identify any intersecting query subsets across setups.

Figure 6 shows how many queries are in each of the four clusters on each setup. The number of fast-running queries on each setup varies between 38 (Setup A) and 73 (Setup D). The number of medium-running queries varies between 5 (Setup D) and 26 (Setup A). The number of slow-running queries varies between 1 (Setup D) and 12 (Setup B) and the number of very-slow-running queries varies between 1 (setups B,C, and D) and 4 (Setup A). Across all setups the fast-running category has the most queries. Most significantly, the in-memory setup (D) has 90% of its queries in this category, which suggests that many of these queries are redundant. They do not add any value to the benchmark for evaluating this type of setup. Even Setup A, which has the least number of fast-running queries, shows 38 (47%) in this category. Overall there seem to be too many short running queries in TPC-DS V2.

There are also very few queries in the very-slow category across all setups, suggesting that more queries should be added to evaluating systems with respect to longer, more complex queries. Setup A has four queries in the very-slow category while setups B, C, and D have only one each.

Next, we analyze the individual queries in each of the categories. The fast category, being the largest category, contains the largest intersection of queries across all four setups (22), namely , 19 ,20 ,21 ,22 ,26 ,30, 39, 42, 43, 52, 53, 55, 63, 66, 73, 29, 81, 85, 89, 91, and 98. These queries do not contribute much to differentiating the four different systems as they execute fast on all of them. The other categories do not have an intersecting set of queries. However, there are some that occur in three out of the four systems. For instance,

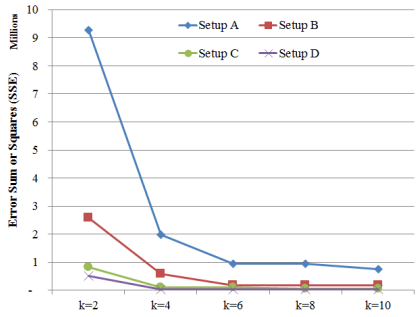


Figure 5: Sum of Squared Errors for each Setup

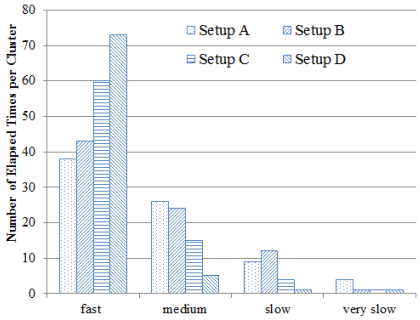


Figure 6: Number of Elements in each Cluster

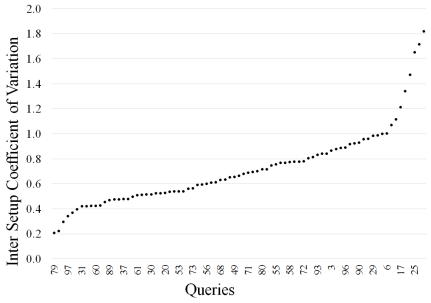


Figure 7: Single-User Normalized Elapsed Times of all Setups

queries 2, 11, 31, and 51 are in the medium category for setups A, B, and C and 76 is in the medium category for setups A, B, and D. Query 4 is in the slow category of setups A, B, and C. Query 78 is in the very-slow category of setups A, B, and C. However, it is in the slow category of Setup D, making Query 78 a common long running query across systems.

Query 78 reports the top customer\item combinations having the highest ratio of store channel sales to all other channel sales. It performs a large join of the sales and returns fact tables of all three sales channels causing 2.8TB of I/O.

4.2.3 Inter Setup Elapsed Times Comparison. The four systems are different in their hardware configuration and general approach to executing queries. Systems A, B, and C use hard disks, with A and B using HDFS, while System D keeps all data in memory. System A and B use identical hardware. To compare the query response times of the four systems in a meaningful way, we first normalize the query response times obtained on one system by the mean of all elapsed times of this system and then use the coefficient of variation to express the disparity of query elapsed time across setups.

Denoting the individual query times on system s during the power run as $T_{P,s}(q_i), 1 \le i \le 99$, we calculate the mean query elapsed time as:

$$\mu_s = \frac{1}{99}\sum_{i=1}^{99} T_{P,s}(q_i), 1 \le i \le 99 \quad (6)$$

We express the normalized response times for each query as:

$$T_{P,s}^{norm}(q_i) = \frac{T_{P,s}(q_i)}{\mu_s}, 1 \le i \le 99 \quad (7)$$

To compare the normalized elapsed times of each query between the four setups, we use the coefficient of variation as a standardized measure of dispersion between the normalized elapsed times. Using the normalized elapsed times for Query q_i on all four setups, we compute the mean query elapsed time across all four setups as:

$$\mu(q_i) = \frac{1}{4}\sum_{s=1}^{4} T_{P,s}^{norm}(q_i) 1 \le i \le 99 \quad (8)$$

Table 3: Elapsed time & data scanned by queries with high normalized elapsed time dispersion

Qry	Elapsed Time[s]				Data Scanned [GB]			
	A	B	C	D	A	B	C	D
12	51	7	4	185	276	4	54	532
16	1508	22	5	11	2219	81	59	217
17	3178	263	89	3	3100	2,021	1547	98
25	3177	76	34	3	3053	215	554	76
77	855	76	19	11	7251	79	543	234
84	482	24	6	70	7	97	76	341
91	16	8	2	22	3	1	19	264
95	619	481	387	2581	1485	1348	1298	37854

we calculate the coefficient of variation function as:

$$CV(q_i) = \sqrt{\frac{1}{99}\sum_{i=1}^{99}(T_{P,s}^{norm}(q_i) - \mu(q_i))^2}, 1 \le i \le 99 \quad (9)$$

The coefficient of variation function CV, plotted in Figure 7, shows the relative dispersion of the normalized elapsed times of the four setups on a ratio scale. The lower the value, the less differ the normalized elapsed times between the four setups, the higher the value the more differ the normalized elapsed times. 90% of all normalized elapsed times vary between 0.2 and 1, which means that 90% of the queries (73 queries) have a relative similar ranking within their data sets when compared across all four systems. This means that the benchmark tests similar characteristics of each of the systems.

There are however, some outliers that have a large coefficient of variation of more than 1, namely, queries 12, 16, 17, 25, 77, 84, 91 and 95. Table 3 shows the elapsed times and amount of data read by these queries. Among these queries, 16, 25, and 84 show the highest dispersion across systems. They have highly selective predicates on the dimension tables, which for Sytems B and C results in relatively low total amount of data read. The high amount of data read by System A in all of the three queries indicates that it cannot take advantage of the highly selective predicates causing it to read over-proportionally more data compared to Systems B and C. Please note that the amount of data scanned for System D, being a columnar organized in-memory solution, includes both memory and disk IO (in case of spills).

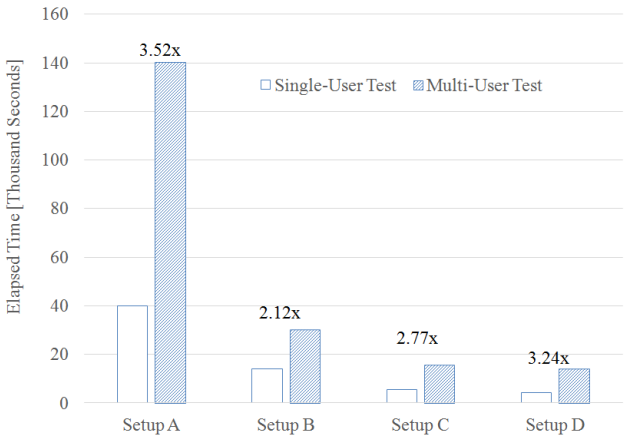


Figure 8: Comparison Single-User/Multi-User tests

4.3 Multi-User Test Analysis

The purpose of the multi-user test is to measure a system's ability to maximize query throughput. Estimating the elapsed time of *n* concurrent sessions by simply multiplying the time of the single-user test with *n*, only works if queries are queued to run consecutively. Unless a single-user test can utilize all systems' resources to 100%, queuing *n* users to a single session will result in sub-optimal performance. Hence, the multi-user test is a very valuable component of a DS benchmark. The challenges running queries in multiple concurrent sessions on one system are different from those running queries in a single session. To avoid over and under-allocation of system resources, which may result in query failure or sub-optimal performance, resources need to be allocated to each session considering all queries currently being executed.

In our multi-user test, a system runs four concurrent sessions each executing 99 queries consecutively. We chose four concurrent sessions because that is the minimum number of concurrent sessions in a valid TPC-DS V2 run. Each session executes the queries in a different permutation to avoid cross-session caching effects. All queries are executed using intra-query parallelism and no query queuing is enabled. The three setups benefit differently from the multi-user run because of the systems ability to schedule concurrent tasks and overlap resources (e.g. IO and CPU). Next, we discuss the CPU and IO resource utilization over time.

Figure 9 displays the CPU and I/O utilization during the single-user run of Setup A. The execution is dominated by the IO (gray graph) as the system is almost 100% IO (black graph) bound during the entire execution of the run. This is in-line with the IO analysis done in Figure 3 of Section 4.1. As this system seems to have the least capability of pruning unnecessary data, it relies heavily on full table scans. The resource utilization for System A during a multi-user run is shown in Figure 10. The IO and CPU graphs in this figure look very similar compared to their counterparts in the single-user figure. However, a closer look shows that the IO utilization and the CPU utilization is slightly higher in both graphs suggesting that the system was able to take advantage of the additional parallelism during the multi-user run. This system remains IO bound and executes the multi-user test in 3.52 times the single-user test (see Figure 10).

Figure 11 displays the CPU and I/O utilization during the single-user run of Setup B. The black graph shows percent CPU used, while the gray graph shows percent I/O used. Both graphs are very ragged. While the CPU graph frequently hits 100%, the I/O graph barely makes it over 50% many times dropping to zero. Notably, there is a period between 10,000 and 11,000 seconds during which the CPU consumption fluctuates vastly. Summarizing the single-user run of Setup B we can say that Setup B uses on average 65% of the available CPU and 32% of the available I/O capacity of the system. At no time interval is the system I/O bound.

Figure 12 shows the corresponding multi-user run. It behaves very differently from the single-user run. Firstly, I/O is the predominant resource hovering around 80%, while CPU varies between 10 and 60%, rarely reaching 100%. The graphs of the multi-user run fluctuate much less compared to the single-user run. Summarizing the multi-user run of Setup B we see that uses an average of 30% CPU and 77% of I/O. To our surprise the numbers for average CPU and I/O utilization are almost flipped in these two cases. The multi-user run was not anymore CPU bound as was the single-user run, but was 7% I/O bound.

Figures 13 and 14 show the corresponding resource utilizations for the single- and multi-user runs of Setup C. This setup behaves similar to Setup B as both IO and CPU graphs are very ragged. The setup is very IO dominated, although it only reaches 100% of IO during very brief periods in the single-user run. In the multi-user run both IO and CPU graphs increase substantially suggesting that the system can take advantage of the increased parallelism.

Figures 15 and 16 show the CPU and IO resource utilization of the in-memory system. In the single-user test this system is CPU bound almost during the entire test, while it shows moderate IO behavior, mostly due to spilling effects of large sort and join operations. The multi-user test looks very similar. Both the IO and CPU graphs are smoother as there is increased concurrency.

4.3.1 Comparing Single-User and Multi-User Tests. Figure 8 compares the single- and multi-user elapsed times for each system. The multi-user test executes the fastest on Setup D (2h 27m 26s), followed by Setup C (4h 18m 36s) and Setup B (8h 18m 14s). The ratio of the elapsed times of the multi- and single-user tests for system *s* is denoted as $R(s)_s = \frac{T_{TT}}{T_{PT}}$. It shows how well a system can absorb concurrent users. A large ratio indicates that a system cannot absorb more users while a small ratio indicates that the system can absorb more users. In the extreme cases: A ratio of 1 means that single and multi-user runs execute in the same time indicating that resources are available for 3 more users. This could mean that the system cannot fully utilize a system with just one user; A ratio of 4 means that the system executes the mutli-user run 4 times that of the single-user run indicating that it has no resources available for the additional 3 users, which in turn could mean that the system is able to fully utilize the system with just one user. These ratios are printed on top of each pair of single- and multi-user tests in Figure 8. Setup A has a ratio of 3.52, Setup B of 2.12, Setup C of 2.77, and Setup D of 3.24. This indicates that systems B and C gain the most and System A the least by running multiple concurrent users. To conclude, these ratios by themselves

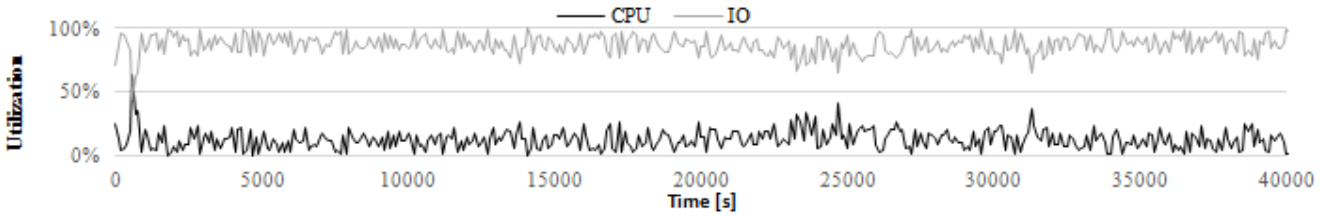


Figure 9: Setup A - CPU and IO Utilization for the Single-User Test

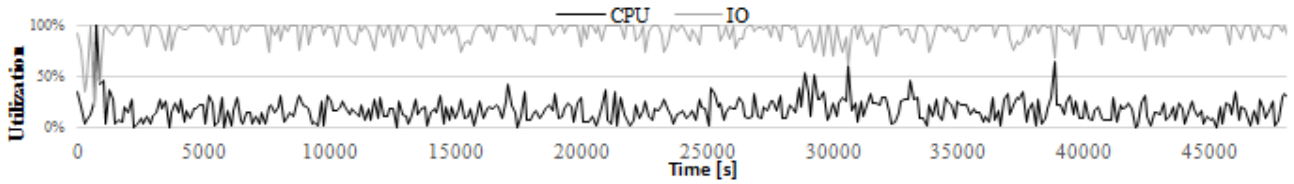


Figure 10: Setup A - CPU and IO Utilization for the Multi-User Test

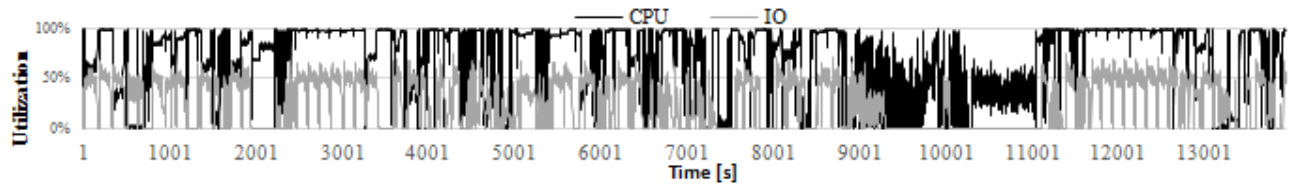


Figure 11: Setup B - CPU and IO Utilization for the Single-User Test

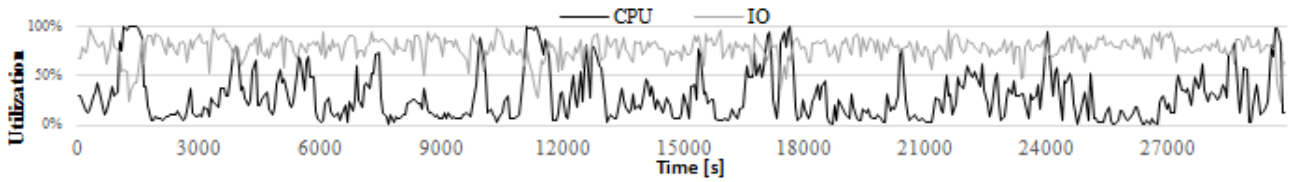


Figure 12: Setup B - CPU and IO Utilization for the Multi-User Test

are, however, no absolute indication of good multi-user performance. A low ratio can be caused by a disproportionately longer single user test or by a disproportionately shorter multi-user test - the latter being more desirable. Similarly, a high ratio can be caused by a disproportionately longer multi-user test or by a disproportionately shorter single-user test - again, the later being more desirable.

4.4 Resource Utilization Analysis

Each decision support query has its own hardware resource utilization pattern, unique to the way it is executed on a particular system in a given situation. A hash-join has a different query pattern than an index-driven join. When comparing systems used in TPC-H publications, it becomes apparent that the system resources most important to decision support queries, especially when dealing with large clusters, are CPU, reads and writes from/to disk, inter-node communication network, and memory. In order to understand a workload it is essential to examine queries that exhibit extreme behaviors, such as CPU intensive queries or I/O intensive queries, and at the same time exhibit a simple structure. These queries enable system analysis along single resource dimensions. This section characterizes the TPC-DS V2 tests according to four resources: (i) CPU, (ii) I/O, (iii) memory, and (iv) network. The sets in Figures 17-19 correlate the I/O, memory, and network utilization to CPU utilization of all TPC-DS V2 queries.

For each resource $r \in \{CPU, IO, Memory, Network\}$ and each one-second time interval during the execution of Workload $w \in \{Power, Throughput, Q_1, .., Q_{99}\}$ on System $s \in \{A, B, C, D\}$, we record how many units of r are used. The unit for CPU is [*thread*], i.e., hardware threads of a huperthreaded x86 processor, and includes user and system time as reported by tools, such as *vmstat*. The unit for IO is [*MBytes/s*] and includes all read and write operations to all disks attached to the system as reported by tools such as *iostat*. The unit for memory is [*GB*] and includes all physical memory allocated by all processes on the system including the file system cache. The

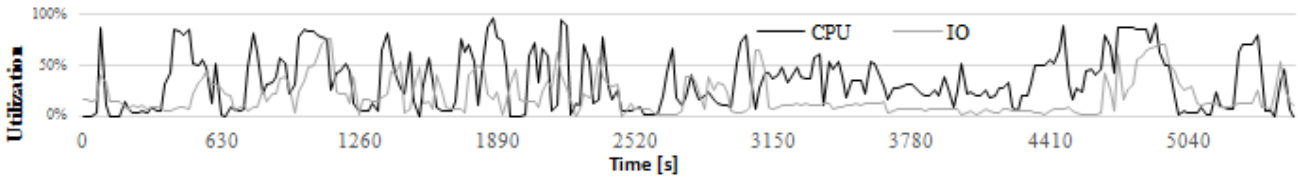


Figure 13: Setup C - CPU and IO Utilization for the Single-User Test

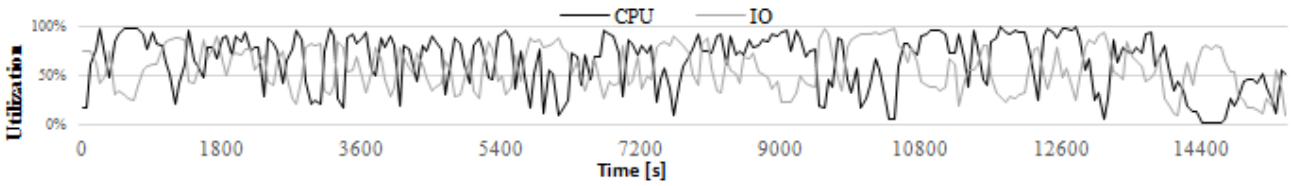


Figure 14: Setup C - CPU and IO Utilization for the Multi-User Test

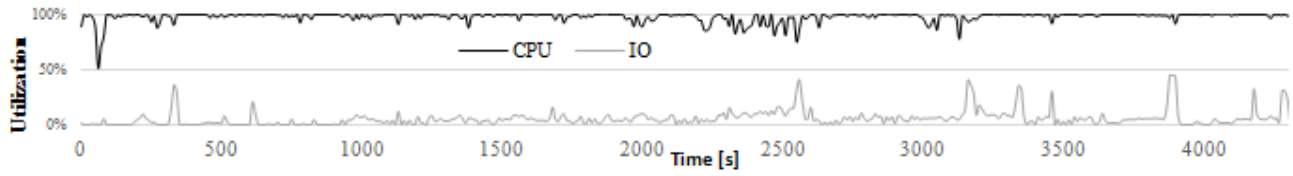


Figure 15: Setup D - CPU and IO Utilization for the Single-User Test

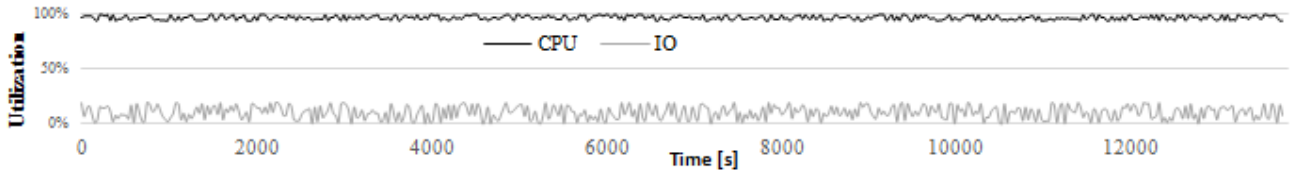
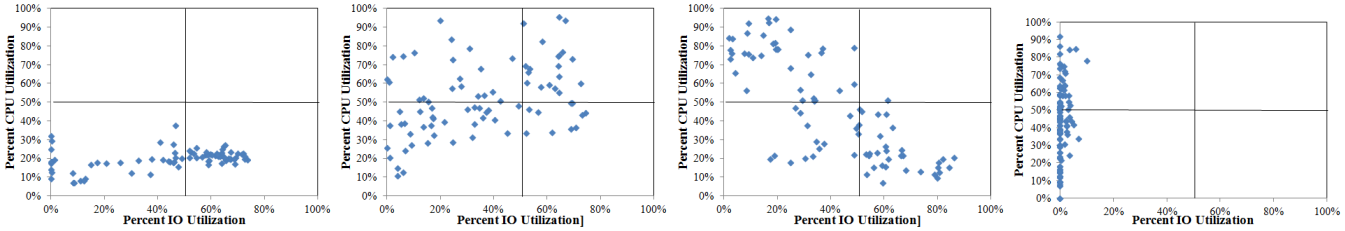


Figure 16: Setup D - CPU and IO Utilization for the Multi-User Test

unit for network is the amount of [*Mbytes/s*] sent and received over the network measured with tools such as netstat. We further record the query elapsed time of this workload as $T_w(s, w)$. We denote the total number of CPU threads available as $U_{max}(CPU, s)$, the maximum I/O available as $U_{max}(I/O, s)$, the total memory available as $U_{max}(MEM, s)$ and the total network bandwidth available as $U_{max}(NET, s)$. Assuming these resources are available to the big data application during the entire duration of the query run, the average resource utilization U_{avg} of Resource r on System $s \in \{A, B, C\}$ for each Workload w, expressed in percent, can be computed as the sum of the ratios of actual resource utilization and maximum resource available during each Time Interval t.

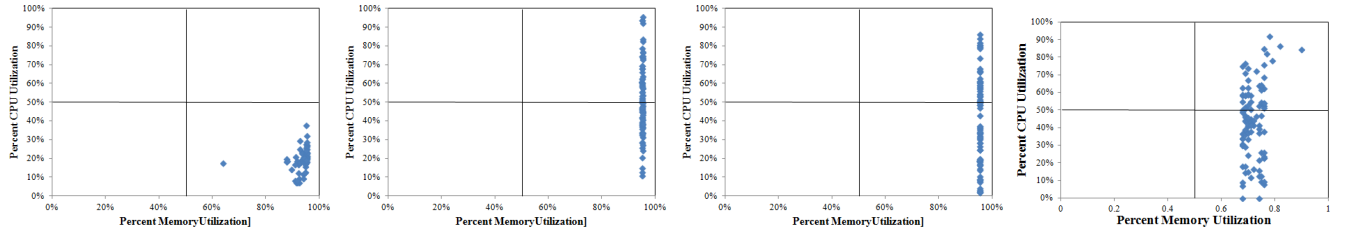
$$U_{avg}(r, s, w) = \frac{100}{T(s, q)} \sum_{t=1}^{T(s,w)} \frac{U_{measured}(r, s, w, t)}{U_{max}(r, s)} \qquad (10)$$

When plotting the resource consumption of two resources R_1 and R_2 w add horizontal and vertical lines, dividing the space into four equally sized quadrants. Queries in the first quadrant, upper right, use both resources R_1 and R_2 intensively, queries located in the second quadrant, upper left, use mostly R_2, queries in the third quadrant, left bottom, use neither R_1 nor R_2 intensively, and queries in the fourth quadrant, right bottom, use both resources intensively. While it is common to divide a plane into four quadrants with the origin point (0,0) being in the middle, the division into four quadrants at 50% resource utilization is our own method to describe query behavior. Because decision support queries are complex and often join multiple tables requiring different join methods, sort large amounts of data and compute aggregate data, their execution pattern is typically not in a steady state for a long time. For example, the usual execution pattern of a hash join can be described as a relatively short burst of intensive I/O during the creation of the hash table of the left side of the join followed by a longer, usually CPU bound phase where the right side of the join is scanned and probed into the hash table. Hence, one cannot infer any specific CPU/ I/O pattern from the two parameters described above. However, these parameters provides a



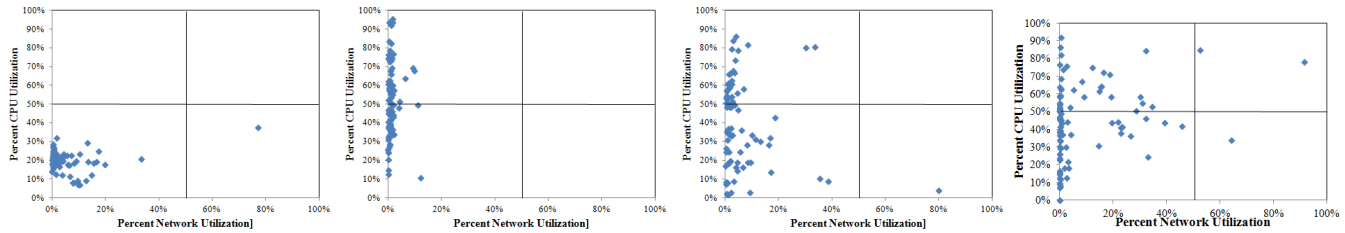
(a) System A (b) System B (c) System C (d) System D

Figure 17: IO-CPU Utilization



(a) System A (b) System B (c) System C (d) System D

Figure 18: Memory-CPU Utilization



(a) System A (b) System B (c) System C (d) System D

Figure 19: Network-CPU Utilization

general idea about resource intensive queries and what the spread of the entire TPC-DS V2 query set is with regard to two resources.

4.4.1 Single-User Resource Utilization. On each system we run the above query subset in a single-user fashion, recording individual query elapsed and resource utilizations.

CPU vs. I/O: Figures 17a, 17b, 17c, and 17d plot the average I/O utilization on the horizontal axis against the average CPU utilization on the vertical axis. On System A, queries fall only into Quadrants III and IV indicating that they are mostly I/O intensive. Most queries show around 20% CPU utilization in both quadrants. 45 queries are located in Quadrant IV. Of the 36 queries that fall in Quadrant III, 10 show less than 5% I/O utilization. On System B, queries fall into all four quadrants: 24 fall into Quadrant I, 25 fall into Quadrant II, 41 fall into Quadrant III and 9 fall into Quadrant IV. On System C, most queries fall into Quadrants II (42) and IV (44). Quadrant III has 12 queries while Quadrant I has only one query. On System D, all queries fall into Quadrant I (43) and Quadrant III (56) indicating that they range from low CPU intensive to high CPU intensive, but using very little IO. This is not surprising as Setup D is the in-memory setup, which uses IO only for the spilling of large sort and join operations.

Query 22 is a low IO/high CPU intensive query on all three system. On Systems B and C, Query 22 uses on average 62% and 85%, respectively, of the available compute power, placing it in Quadrant II. On System A it uses only 32% of the available CPU power, placing it in III quadrant. While 32% CPU utilization is generally low, on this system it is the query with the highest CPU consumption. Query 22 uses the compute intensive *rollup* functionality in SQL. For each product name, brand, class, and category, it calculates the average quantity on hand rolling-up data by product name, brand, class, and category. Query 22 only has to scan 1% (21GB) of the total dataset. Query 88 can be classified as a high IO/low CPU intensive query on all three systems. It is located in Quadrant IV utilizing on average between 70% (System A) and 84% (System C) of the

available IO bandwidth. Query 88 analyzes store sales by returning in one row the number of sales occurring during each 30 minute time interval between 8:30am and 12:30pm. As a consequence, this query, without any optimization, scans *store_sales* eight times to a total of 5,161 GB.

CPU vs. Memory: Figures 18a, 18b, 18c, and 18d plot the average memory utilization on the horizontal axis against the average CPU utilization on the vertical axis for each system. Queries are very memory intensive across all four systems. Most queries are using between 90% and 95% of the available memory on Setups A, B, and C. Setup B and C seem to pre-allocate and never release memory as all queries use about 95% of the available memory. System C shows some variation of memory utilization with most queries using between 88% and 95% of the available memory with one outlier using only 64%. The in-memory Setup D shows between 70% and 90% memory utilization. It is not surprising that this setup does not use 100% of the memory as this is a very large memory system and not all memory was used for the workload. While most queries seem to use between 70% and 80% of the available memory, the system used for Setup D seem to release some memory. Likely the memory that is used to store the actual data is fixed and the memory used for query execution is partly released between queries.

CPU vs. Network: Figures 19a, 19b, 19c, and 19d plot the average network utilization on the horizontal axis against the average CPU utilization on the vertical axis for each system. All four setups show a very different network utilization. Except for two outliers at 32% and 80%, most queries run on Setup A show only up to 20% network utilization and are located in Quadrant IV. Network utilization is the lowest on Setup B where most queries use less than 5% of the network's bandwidth. Four queries use around 10%. Setups C and D have the most wide-spread network utilization. Most queries use less then 20%, four use between 30% and 40% and one uses 80%. There is no common pattern to be found among these different systems indicating that they distribute queries operations for the various queries very differently.

To conclude we can say that many queries show a wide spectrum of resource utilization across the four setups indicating that they are very diverse and utilize the systems depending on their architectural peculiarities.

5 CONCLUSION

Our analysis of pivotal parts of the benchmark and our experimental results on four different setups show that TPC-DS V2 stresses many aspects of SQL-based big data systems. Different resource consumptions and elapsed times of the queries suggest that the workload is *discriminatory* enough to reveal interesting characteristics in current big data solutions:

- Our test systems apply various degrees of data pruning, but no system prunes the maximum possible.
- Query elapsed times during single-user runs on each system vary between seconds and hours.
- Our test systems optimize differently for single- and multi-user runs. Some benefit more some less.
- Our test systems show vastly different resource utilizations for single- and multi-user runs.
- Many queries show a wide spectrum of resource utilization across the four setups.

Our analysis also revealed some of the weaknesses of TPC-DS V2. Using elapsed time coefficient of variation analysis and K-means analysis we discovered that query elapsed times are skewed towards short running queries, especially for in-memory systems (Setup D). To the contrary, we identified that only few queries show very long elapsed times on our systems, which suggest that more complex queries should be added to counter balance the short running queries. We also discovered that there is large potential for current systems to prune unnecessary I/O. Plotting I/O, memory and, network consumptions in two dimensional grids against CPU utilization we identified queries that are single resource heavy on all systems. These queries can be used to calibrate systems before running the entire TPC-DS V2 benchmark.

6 ACKNOWLEDGEMENT

This work has been supported through grants by the German Federal Ministry for Education and Research as Berlin Big Data Center (funding mark 01IS14013A), through grants by the European Union's Horizon 2020 research and innovation program under grant agreement 688191 and by the Alexander von Humboldt Foundation. Many thanks to Srikanth Kandula for his valuable comments and suggestions, which have led to significant improvement on the presentation and quality of this paper.

REFERENCES

[1] 2017. IBM Big SQL. https://www.ibm.com/support/knowledgecenter/en/SSPT3X_4.1.0/com.ibm.swg.im.infosphere.biginsights.product.doc/doc/bi_sql_access.html. (2017). IBM.
[2] 2017. Oracle Big Data SQL. https://www.oracle.com/database/big-data-sql/index.html. (2017). Oracle Corporation.
[3] 2017. SAP Vora. https://www.sap.com/products/hana-vora-hadoop.html. (2017). SAP.
[4] AMP Lab. 2013. AMP Lab Big Data Benchmark. (2013). https://amplab.cs.berkeley.edu/benchmark/.
[5] Timothy G Armstrong, Vamsi Ponnekanti, Dhruba Borthakur, and Mark Callaghan. 2013. LinkBench: a Database Benchmark Based on the Facebook Social Graph. In *SIGMOD*. 1185–1196.
[6] Cloudera. [n. d.]. Impala. http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html. ([n. d.]).
[7] Alain Crolotte. 2009. Issues in Benchmark Metric Selection. In *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*. 146–152.
[8] Paul M. Davis. [n. d.]. Pivotal HAWQ Benchmark Demonstrates Up To 21x Faster Performance on Hadoop Queries Than SQL-like Solutions. https://content.pivotal.io/blog/pivotal-hawq-benchmark-demonstrates-up-to-21x-faster-performance-on-hadoop-queries-than-sql-like-solutions. ([n. d.]).
[9] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (2008), 107–113.
[10] Justin Ericson, Greg Rahn, Marcel Kornaker, and Yanpei Chen. [n. d.]. Impala Performance Update: Now Reaching DBMS-Class Speed. http://blog.cloudera.com/blog/2014/01/impala-performance-qdbms-class-speed/. ([n. d.]).
[11] Facebook. [n. d.]. Presto. https://prestodb.io/. ([n. d.]).
[12] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. In *SIGMOD*.
[13] Simon Harris. [n. d.]. Big SQL 3.0: Hadoop-DS benchmark - Performance isnâĂŹt everythingâĂę. https://developer.ibm.com/hadoop/blog/2014/12/02/big-sql-3-0-hadoop-ds-benchmark-performance-isnt-everything/. ([n. d.]).
[14] Hortonworks. [n. d.]. Stinger. http://hortonworks.com/innovation/stinger/. ([n. d.]).
[15] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis. In *ICDEW*.
[16] Kiyoung Kim, Kyungho Jeon, Hyuck Han, Shin gyu Kim, Hyungsoo Jung, and H.Y. Yeom. 2008. MRBench: A Benchmark for MapReduce Framework. In

Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on. 11–18.

[17] Min Li, Jian Tan, Yandong Wang, Li Zhang, and Valentina Salapura. 2015. SparkBench: A Comprehensive Benchmarking Suite for in Memory Data Analytic Platform Spark. In *Proceedings of the 12th ACM International Conference on Computing Frontiers (CF '15)*. ACM, New York, NY, USA, Article 53, 8 pages.

[18] J. MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. University of California Press, Berkeley, Calif., 281–297. http://projecteuclid.org/euclid.bsmsp/1200512992

[19] Michael Armbrust, Zongheng Yang. [n. d.]. Exciting Performance Improvements on the Horizon for Spark SQL. https://databricks.com/blog/2014/06/02/exciting-performance-improvements-on-the-horizon-for-spark-sql.html. ([n. d.]).

[20] Richard C Murphy, Kyle B Wheeler, Brian W Barrett, and James A Ang. 2010. Introducing the Graph 500. *Cray UserâĂŹs Group (CUG)* (2010).

[21] Raghunath Othayoth Nambiar and Meikel Poess. 2006. The Making of TPC-DS. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*. 1049–1058.

[22] Raghunath Othayoth Nambiar, Meikel Poess, Akon Dey, Paul Cao, Tariq Magdon-Ismail, Da Qi Ren, and Andrew Bond. 2014. Introducing TPCx-HS: The First Industry Standard for Benchmarking Big Data Systems. In *Performance Characterization and Benchmarking. Traditional to Big Data - 6th TPC Technology Conference, TPCTC 2014, Hangzhou, China, September 1-5, 2014. Revised Selected Papers*. 1–12.

[23] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. 2008. Pig Latin: A Not-so-Foreign Language for Data Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*. 1099–1110.

[24] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. 2009. A Comparison of Approaches to Large-Scale Data Analysis. In *SIGMOD*. 165–178.

[25] Meikel Pöss, Raghunath Othayoth Nambiar, and David Walrath. 2007. Why You Should Run TPC-DS: A Workload Analysis. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*. 1138–1149.

[26] Tilmann Rabl, Ahmad Ghazal, Minqing Hu, Alain Crolotte, Francois Raab, Meikel Poess, and Hans-Arno Jacobsen. 2014. BigBench Specification V0.1. In *Specifying Big Data Benchmarks*, Tilmann Rabl, Meikel Poess, Chaitanya Baru, and Hans-Arno Jacobsen (Eds.). Lecture Notes in Computer Science, Vol. 8163. Springer Berlin Heidelberg, 164–201.

[27] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. 2009. Hive: A Warehousing Solution Over a Map-Reduce Framework. *PVLDB* 2, 2 (2009), 1626–1629.

[28] Robert Tibshirani, Guenther Walther, and Trevor Hastie. 2001. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63, 2 (2001), 411–423. https://doi.org/10.1111/1467-9868.00293

[29] Transaction Processing Performance Council. 2015. Specification TPC-DS Version 2.1. http://www.tpc.org/tpcds/default.asp. (2015).

[30] Transaction Processing Performance Council. 2016. Top 10 TPC-H publications grouped by scale factor. (07 2016). http://www.tpc.org/tpch/results/tpch_perf_results.asp

[31] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, and E. Baldeschwieler. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, ACM (Ed.). 5.

[32] Reynold S Xin, Josh Rosen, Matei Zaharia, Michael J Franklin, Scott Shenker, and Ion Stoica. 2013. Shark: SQL and Rich Analytics at Scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data*. ACM, 13–24.