Meikel Pöss

Methodologies for a Comprehensive Approach to Measuring the Performance of Decision Support Systems

Technische Universität München

Technische Universität München

Fakultät für Informatik

Methodologies for a Comprehensive Approach to Measuring the Performance of Decision Support Systems

Meikel Pöss

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende(r): Prof. Dr. Michael Gerndt

Prüfer der Dissertation:

1. Prof. Dr. Hans-Arno Jacobsen
2. Prof. Oliver Günther, Ph.D.

Die Dissertation wurde am 29.11.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 12.12.2017 angenommen.

Abstract

Industry standard benchmarks for measuring the performance of computer systems have been very valuable tools for both companies providing solutions and customers seeking the best solution for their needs. Solution providers utilize benchmarks for internal and external purposes. Internally, they gauge the performance of current features and drive the development of new features. Externally, they are used in official benchmark publications to claim superior performance and price-performance compared to competitors. Customers seeking solutions for their business problem can be confident that publications of Industry standard benchmarks accurately reflect system performance.

Because of these needs, benchmark consortia such as the Standard Performance Evaluation Corporation (SPEC) [1], Storage Performance Council (SPC) [2], Embedded Microprocessor Benchmark Consortium (EMBC) [3], and the Transaction Processing Performance Council (TPC) [4] continue to develop benchmarks.

Since the early decision support systems (DSS) were developed in the 1960s, they have evolved over time, from relying on isolated, specialized applications to using connected, general purpose database management systems. The DSS, combining data from a myriad of data sources, triggered the development of data integration systems. With the evolution and broad availability of network technology, the application of DSS is extended into the Internet of Things (IoT).

In this work, we propose methodologies for a comprehensive approach to measuring the performance of decision support systems in the domains of IoT, data integration (DI) and SQL-based big data systems. Firstly, we develop a benchmark for measuring the performance of IoT gateway systems that must ingest, persist, and analyze massive amounts of data arriving from edge devices. This benchmark was integrated into TPCx-IoT, the first industry standard benchmark to measure the performance of IoT gateway systems. It was released by the TPC in September 2017. Secondly, we develop a data integration benchmark for measuring the performance of systems while integrating data from a variety of data source, formatting it into a unified data model representation, and loading it into a data warehouse. This benchmark was integrated into the first industry standard benchmark for measuring the performance of data integration systems, TPC-DI, and released by the TPC in January 2014. Thirdly, we develop changes to be implemented in the first version of the TPC-DS, which will enable big data systems to run the benchmark. These changes were incorporated into the second version of TPC-DS, released by the TPC in August 2015.

For each of the benchmarks, TPCx-IoT, TPC-DI, and TPC-DS, we conduct a performance analysis to demonstrate the benchmark measure performance of DSS in the environment they address.

Zusammenfassung

Für Hersteller von Computersystemen, sowie für deren Kunden sind Industriebenchmarks zur Leistungsmessung wertvolle Hilfsmittel. Für die Hersteller von Computersystemen sind Industriebenchmarks von internem und externem Nutzen. Intern messen sie mit ihnen die Leistung neuer Funktionalitäten ihrer Computersysteme, und haben damit die Möglichkeit noch während der Produktentwicklung Leistungsdefizite zu erkennen und Leistungsverbesserungen durchzuführen. Extern sind sie ein Marketinginstrument für den direkten Leistungs- und Preis/Leistungsvergleich mit den Computersystemen ihrer Konkurrenten. Für Kunden sind Industriebenchmarks ein verlässliches Werkzeug zum Leistungs- und Preis/Leistungsvergleich der Computersysteme von unterschiedlichen Herstellern.

Um die Bedürfnisse von Herstellern und Kunden zu decken arbeiten seit Ende der achtziger Jahre Industriebenchmark Konsortien, wie zum Beispiel die Standard Performance Evaluation Corporation (SPEC) [1], Storage Performance Council (SPC) [2], Embedded Microprocessor Benchmark Consortium (EMBC) [3], und die Transaction Processing Performance Council (TPC) [4] ständig an der Weiterentwicklung ihrer existierenden Benchmarks und an Neuentwicklungen für aufkommende Anwendungsgebiete.

Die ersten Decision Support Systeme (DSS) aus den sechziger Jahren wurden üblicherweise als isolierte Spezialentwicklungen implementiert. In den darauf folgenden Jahren kamen immer häufiger Standardsoftware, wie zum Beispiel relationale Datenbanksysteme, zur Implementierung von DSS zum Einsatz. In den letzten Jahren wird verstärkt Open-Source Software eingesetzt. Die für die Entscheidungsfindung in Unternehmen benötigten Daten kommen oft aus verschiedenen Unternehmensbereichen, die ihre Daten in eigenen Systemen und Formaten speichern. Diese Daten zeitig und effizient in ein einheitliches Datenmodel zu integrieren wird eine immer wichtiger werdende Aufgabe von modernen DSS. Mit der Entwicklung und Verbreitung von immer schnelleren Netzwerktechnologien eröffnet sich die Möglichkeit, DSS in Internet of Things (IoT) Anwendungen zu integrieren.

In dieser Arbeit werden Methoden für einen umfassenden Ansatz zur Leistungsmessung von DSS in den Bereichen Internet of Things (IoT), Datenintegration (DI) und

SQL-basierten Big Data Systemen entwickelt. IoT Gateway Systeme sind dadurch geprägt, dass sie große Datenmengen von vielen Sensoren entgegennehmen, speichern und in Realzeit analysieren müssen. In dieser Arbeit werden zuerst Methoden für die Leistungsmessung von IoT-Gateway-Systeme entwickelt, die in TPCx-IoT, dem ersten Industriebenchmark zur Leistungsmessung von IoT-Gateway-Systemen, standardisiert wurden. Die TPC hat TPCx-IoT im September 2017 ratifiziert.

Datenintegrationssysteme sind dadurch geprägt, dass sie Daten von einer Vielzahl von Datenformaten in ein einheitliches Datenmodell überführen, und diese Daten dann in ein Datenbanksystem, zum Beispiel ein Data Warehouse, laden in dem sie verwaltet und schnell analysiert werden können. In dieser Arbeit werden Methoden zur Leistungsanalyse von Datenintegrationssystemen entwickelt, die in dem ersten Industriebenchmark zur Leistungsmessung von Datenintegrationssystemen, TPC-DI, standardisiert wurden. Die TPC hat TPC-DI im Januar 2014 ratifiziert.

Sind alle Daten in ein zentrales Datenbanksystem integriert, dann greifen verschiedene Applikationen mit Anfragen auf die Daten zu. Diese Anfragen werden in vielen Systemen mittels der Anfragesprache Structured Query Language (SQL) gestellt. Für die Leistungsmessung von SQL basierten Systemen existiert der Industriebenchmark TPC-DS. Da in den letzten Jahren immer häufiger SQL basierte Big Data Systeme als Datenbanksystem für DSS eingesetzt werden, diese aber nicht in der Lage sind TPC-DS auszuführen, werden in dieser Arbeit Änderungen erarbeitet, die es SQL basierten Big Data Systemen ermöglicht, TPC-DS durchzuführen. Diese Änderungen wurden in der zweiten Version des TPC-DS Industriebenchmarks standardisiert. TPC-DS Version 2 wurde im August 2015 von der TPC ratifiziert.

Für jeden der oben genannten Benchmarks, TPCx-IoT, TPC-DI, und TPC-DS werden Leistungsanalysen auf verschiedenen Hard-und Softwareplatformen durchgeführt, um zu demonstrieren wie diese Industriebenchmarks die Leistung von DSS in den verschiedenen Umgebungen messen.

Acknowledgments

This doctoral dissertation was prepared at the Department of Informatics of the Technical University of Munich under the supervision of Prof. Dr. Hans-Arno Jacobsen. I would like to express my sincere gratitude to my supervisor Prof. Dr. Hans-Arno Jacobsen and mentor, Prof. Dr. Tilmann Rabl, for their continuous support and guidance.

I would like to thank the rest of my thesis committee: Prof. Dr. Oliver Günther for agreeing to be the second examiner, and Prof. Dr. Michael Gerndt for accepting to chair the committee.

I would like to thank Raghunath Nambiar, Chinmayi Narasimhadevara, and Karthik Kulkarni from Cisco for their valuable information about the Internet of Things and providing hardware to collect performance data running TPCx-IoT on industry grade gateway systems.

Advice given by the following Transaction Processing Performance Council representatives has been of great help in the completion of these benchmarks: Michael Brey, Charles Levine, Alain Crolotte, Lubor Kollar, Murali Krishna, Bob Lane, Larry Lutz, Juergen Mueller, Bob Murphy, Doug Nelson, Bryan Smith, Eric Speed, Jack Stephens, John Susag, Dave Walrath, Mike Nikolaiev, Francois Raab, Yeye He, Jerry Buggert, Nicholas Wakou, Vince Carbone, Wayne Smith, Yanpei Chen, John Poelman, and Seetha Lakshmi.

I am particularly grateful to my wife, Monica Celle-Küchenmeister. Her unconditional love, support, encouragement, proof-reading and, foremost, her belief in me made my dream come true. Along with her, I would like to acknowledge my three children, Darius, Dehlia, and Dorian. They are my greatest source of love and relief from scholarly endeavors.

Finally, I would like to thank my parents. They have always allowed me to pursue my dreams. It is not always been easy to let your only child live so far away. But without your blessings and continued moral support, I could not have undertaken this big step.

Contents

Chapter 1

Introduction

The origins of decision support systems (DSS) go as far back as the 1960s when model-oriented DSS were developed. At that time, the emphasis was on building management information systems as "integrated man/machine systems to provide information for supporting the operations, management, and decision-making functions in an organization" [5]. Decision support systems have evolved over time from relying on isolated, specialized applications to using connected, general purpose database management systems (DBMS). The ability to integrate data sources from a myriad of DBMS into one decision support system triggered the development of data integration systems. With network technology evolving and becoming broadly available, the application of decision support systems has been extended to the Internet of Things (IoT).

This work addresses the need for methodologies for a comprehensive approach to measuring the performance of decision support systems in the areas of IoT, data integration (DI) and SQL-based big data systems. Firstly, we developed a benchmark to measure the performance of IoT systems that must ingest, persist, and analyze massive amounts of data arriving from edge devices. This benchmark was integrated into TPCx-IoT [6], the first industry standard benchmark for measuring the performance of IoT gateway systems. It was released by the TPC in September 2017. Secondly, we developed a data integration benchmark for measuring the performance of systems that integrate data from a variety of data source formats into a unified data model representation and load it into a data warehouse. This benchmark was integrated into TPC-DI [7], the first industry standard benchmark to measure the performance of data integration. It was released by the TPC in January 2014. Thirdly, we developed changes in the first version of TPC-DS that were necessary for big data systems to run the benchmark. These changes were incorporated into the second version of TPC-DS [8] in August 2015. TPC-DS Version 2 is the first industry standard benchmark for measuring the performance of SQL-based big data systems.

1.1 Motivation

In the late 1980s, unverifiable marketing claims about system performance by hardware vendors led to the formation of industry standard benchmark consortia [9] such as the Standard Performance Evaluation Corporation (SPEC) [1] and the Transaction Processing Performance Council (TPC) [4]. By enabling all system vendors to compete on a level playing field, these consortia allow for fair performance comparisons amongst vendors.

Recognizing the need for a standard benchmark to measure the performance of DSS systems, the TPC released its first data warehouse benchmark, TPC-D [10], in April 1994. For the technology available at that time, TPC-D imposed many challenges to both hardware and DBMSs. The development of aggregate/summary structures in many products at the end of the 90's, which dramatically decreased query elapsed times resulting in an over-proportional increase in TPC-D's main performance metric, effectively broke the benchmark. While working on a long term solution, its next generation decision support benchmark (TPC-DS) [11, 12, 13], the TPC spun off two modified versions of TPC-D, namely TPC-H and TPC-R in April 1999 [14].

Since TPC-DS was released in November 2012, the development of new decision support technology has accelerated, and its use has broadened dramatically. Big data platforms enable decision support deployments on massive scales; this triggered the need to integrate data from a myriad of data source. Lately, the wide proliferation of networks and low cost sensor technologies have enabled the integration of data from hundreds of thousands of devices into continuous data streams to DSS.

Without having industry standard benchmarks for measuring the performance of DSS in these complex environments, system vendors can make marketing claims based on unverifiable performance numbers [15, 16, 17, 18, 19, 20, 21, 22, 23]. This kind of "benchmarketing" is not new to the industry, and it is precisely what triggered the founding of the TPC 25 years ago.

Industry standard benchmarks provide the tools and rule framework for both companies providing solutions and customers seeking the best solution for their needs. Solution providers utilize benchmarks for internal and external purposes. Internally, they gauge the performance of current features and drive the development of new features. Externally, they are used in official benchmark publications to claim superior performance and price-performance compared to competitors. Customers seeking solutions for their business have confidence that these publications will provide a fair performance comparison of products from various solution providers.

1.1.1 Motivation for an industry standard benchmark for IoT systems (TPCx-IoT)

According to a 2017 projection by Gartner [24], the total number of IoT devices will more than double between now and 2020 (8.4 Billion to 20.4 Billion). While the initial hype around this Internet of Things (IoT) stems from consumer use cases, the number of devices and data from enterprise use cases is significant in terms of market share. With companies being challenged to choose the right digital infrastructure from different providers, there is a pressing need to objectively measure hardware, operating system, data storage, and data management systems that can ingest, persist, and process massive amounts of data arriving from sensors (edge devices).

1.1.2 Motivation for an industry standard benchmark for data integration systems (TPC-DI)

Historically, the process of synchronizing a decision support system with data from operational systems has been referred to as Extract, Transform, Load (ETL), and the tools supporting such process have been referred to as ETL tools. Recently, ETL was replaced by a more comprehensive acronym, data integration (DI). DI describes the process of extracting and combining data from a variety of data source formats, transforming that data into a unified data model representation and loading it into a data warehouse. This is done in the context of various scenarios, such as data acquisition for business intelligence, analytics and data warehousing, but also synchronization of data between operational applications, data migration and conversions, master data management, enterprise data sharing and delivery of data services in a service-oriented architecture context amongst others. With these scenarios relying on up-to-date information, it is critical to implement a high-performing, scalable and easy to maintain data integration system. This is especially important as the complexity, variety and volume of data is constantly increasing and performance of data integration systems is becoming very critical. Despite the significance of a high- performing DI system, there has been no industry standard for measuring and comparing their performance.

As a consequence, vendors publish one-off benchmark results claiming “World Record” performance [19, 20, 21, 22]. Many of the vendors use simplistic data-sets, for instance, TPC-H, that were not developed for ETL and cannot be used to realistically measure the performance of ETL systems. Hence, there is no mechanism for comparing performance results. The TPC, acknowledging this void, has released TPC-DI, an innovative benchmark for data integration.

1.1.3 Motivation for an industry standard benchmark for SQL-based big data systems (TPC-DS V2)

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 has begun to resurrect in many of these solutions, e.g., Hive, Stinger, Impala, Shark, and Presto. At the same time, RDBMS vendors are adding Hadoop support to their SQL engines, e.g., IBM's Big SQL, Actian's Vortex, Oracle's Big Data SQL, and SAP's HANA.

In need of benchmarks to showcase the performance of their big data SQL engines, many vendors used custom benchmarks that they derived from the first version of the TPC's 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 [15, 16, 17, 18].

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.

1.2 Problem Statement

The advent of big data platforms, the proliferation of networks and the availability of low cost sensor technologies expand the use of DSS into many areas. At the same time, DSS solution providers require methodologies for measuring the performance of their solutions, and customers seeking products to solve their DSS problem require reliable performance data for making informed purchase decisions. This work divides the problem of measuring the performance of DSS into three main research areas:

1. *Ingesting data from numerous sensors at high frequency and performing real-time analytics for device monitoring*
2. *Integrating data from a myriad of data sources into a DSS for subsequent querying*
3. *Performing the initial load, incremental periodic data maintenance and running complex analytic queries in single- and multi-user situations of SQL-based big data systems*

Each of the three areas listed above have their own requirements for the basic components of a benchmark: (i) *data generation*, (ii) *execution rules*, (iii) *workload*, and (iv) *metric*. Data ingested from sensors is simple, comprising of a time stamp, location and value. It is ingested at high frequencies into the data management system (many samples per second). Data integrated from a myriad of data sources is diverse and complex consisting of structured and unstructured data that need to be cleansed. It is loaded at hourly or daily frequencies. Data for DSS, which run complex analytics in back-end systems, is divided into initial load data and incremental load data. Data that is initially loaded fulfills referential integrity, while data that is loaded incrementally does not. Initial data is loaded very infrequently, that is, once a month or once a year, while incremental data is loaded on an hourly or daily basis.

The execution rules, workload and metric are intrinsically connected to each other and they are equally powerful in controlling performance measurements. Execution rules define the way benchmarks are executed, the type of workloads challenge specific functionality of the system being measured and the metrics emphasize those parts of the execution that are timed. When data is ingested at high frequencies for real-time analytics, loading and querying occur concurrently. Benchmarks for these DSS applications must be able to generate many queries at high frequency, while scheduling data ingestion operations. Because the queries being issued are simple in nature, the metric must emphasize data ingestion throughput rather than query elapsed time. The emphasis when measuring the performance of data integration systems is on the integration and load processes with infrequent querying of data. Benchmarks for data integration systems must be able to provide a complex integration workload. The queries executed as part of the data integration process are simple, and metric emphasis is on the data load throughput. The emphasis when measuring the performance of back-end DSS is on querying data as a single user and as multiple concurrent users with very infrequent reloads of the entire data set and periodic data maintenance.

In addition to the above listed general benchmark components, industry standard benchmarks require the following additional components: (i) *audit rules*, (ii) *disclosure rules*, and (iii) *pricing rules*. It is imperative that customers who make purchase decisions based on performance results should be confident that the performance results conform to the rules laid out by the benchmark. The audit rules of industry standard benchmarks define how performance data, obtained during benchmark runs, can be verified. Audit rules are benchmark specific and, therefore, need to be developed for each benchmark. Disclosure rules define what information needs to be disclosed as part of a benchmark publication. They allow for easy comparison of different benchmark results, and allow anybody to verify that the performance data was obtained according to the benchmark rules.

The quality and, ultimately, the success of benchmarks depend on many parameters. Jim Gray defined the following four design principles that have been widely recognized as success criteria for industry standard benchmarks [25]: (i) The benchmark measures key performance criteria of the domain it addresses (*relevant*). (ii) The benchmark can be implemented on many different systems and architectures (*portable*). (iii) The benchmark should scale with computer systems becoming more powerful (*scalable*).

(iv) The benchmark must be understandable (*simple*). Michael Stonebraker suggested that industry standard benchmarks should also be *challenging*, that is, benchmarks should not reflect the least common denominator among the technologies available from the companies that developed the benchmark, but they should challenge all technologies using complex customer problems [26]. Based on his experience in developing successful TPC benchmarks, Karl Huppler further extended this list with three more criteria [27]: (i) The benchmark imposes the same workload every time it is run, resulting in similar performance results (*repeatable*). (ii) Benchmark results are verified and real (*verifiable*). (iii) Vendors are not excluded because of the high cost of benchmarking (*economical*).

1.3 Approach

In order to address the complex problem of providing industries with an industry standard benchmark for measuring the performance of DSS, we divided the problem into three areas, that is, into three separate benchmarks, each of which were standardized into industry standard benchmarks by the TPC:

1. TPCx-IoT: An industry standard benchmark for measuring the performance of IoT gateway systems.
2. TPC-DI: An industry standard benchmark for measuring the performance of data integration systems.
3. TPC-DS Version 2: An industry standard benchmark for measuring the performance of SQL-based big data systems.

1.3.1 TPCx-IoT

To measure the performance of systems ingesting data from numerous sensors at high frequency and performing real-time analytics for device monitoring, we developed the benchmark TPCx-IoT. To grant TPCx-IoT a realistic context, we modeled it after a monitoring gateway for sensor data originating from power substations of a typical electric utility provider. We developed the data generator based on the Yahoo! Cloud Serving Benchmark framework (YCSB) [28]. YCSB is well-known, open source, easily adaptable to specific needs, and its built-in database management interface layer allows connections to many common open-source database management systems including NoSQL DBMS. We modified YCSB in two areas: (i) To make the data set realistic, we add data from an electric utility provider. (ii) Because the real-time analytical queries executed on IoT gateways are time-series range queries, we modified the read workload generator in YCSB to issue range queries instead of scattered reads. The range queries are simple group-by aggregations selecting data for one specific sensor.

As the main performance metric, we designed a throughput metric, IoTps. It reflects the effective gateway ingestion rate in seconds that the system can support during a 30-minute measurement interval, while concurrently executing real-time analytic queries.

Most industry implementations of IoT gateway solutions use commodity hardware, especially x86-based systems and open-source software. Therefore, it seems reasonable to develop TPCx-IoT following TPC's *express benchmark* framework. Express benchmarks are based on predefined executable software kits. They are restricted to the technology implemented in the kit. However, they must provide mechanisms to allow easy extensions to other technologies. Since TPCx-IoT uses the YCSB framework, it already includes a database management interface layer that allows connections to many common open-source technologies.

Because TPCx-IoT is an express benchmark, we include audit mechanism directly into its executable software kit. These methods are divided into pre- and post-run checks. For instance, pre-run checks assure that the kit has not been altered and that the data storage system uses at least a three-way data replication, while post-run checks assure that the benchmark runs for the required time and that the system sustains the minimal average data ingest rate of 20 sensor reads per second.

1.3.2 TPC-DI

To measure the performance of data integration systems, we developed TPC-DI. Data integration systems have no standard language for defining the data flow required for integrating data from various sources. For this reason, we decided to develop TPC-DI following TPC's *enterprise benchmark* framework. This enables us to define a set of functional requirements that can be run on any DI system, regardless of specific hardware or software.

We designed TPC-DI using the data integration processes of a retail brokerage firm that populates a DSS with transformed data from a variety of different sources, including a trading system, internal Human Resource (HR) and Customer Relationship Management (CRM) systems.

We designed the data model of TPC-DI to exercise much of the functionality typically used in DI systems. It consists of two main parts, the *source data model* and the *target data model*. The source data model represents the input data set to the data integration process. We designed it by combining online trading operations with other typical internal data sources, that is, human resource and customer management system, externally acquired data, financial newswire data, and customer prospect data. These sources cover many data formats, for example, comma separate values (CSV), plain text, multi-format, and XML. The target data model resembles a dimensional DSS following common practice in the industry [29]. It consists of multiple fact tables that share various types of dimensions. This snowflake schema variant enables easy and efficient responses to typical business questions asked in the context of a retail brokerage firm.

To enable both historical loads, that is, loads that occur during the initial creation of a DSS and periodic incremental updates to a DSS, we implemented TPC-DI with two different execution phases, a historical load phase and two incremental load phases. Splitting the execution into these two phases is important because they have very different performance characteristics and impose different requirements on the DSS as it does not need to be queryable during the historical load, but it does need to be queryable during each incremental load. For each phase, we calculate the rows per second throughputs. We then combine the two throughput numbers into one main performance metric using the geometric mean of the two.

It represents the number of rows processed per second as the geometric mean of the historical and incremental loading phases. In order to calculate throughput numbers, we need to define the measurement interval and what we mean by rows processed. TPC-DI defines four completion time stamps (CTs) to be taken at a precision of 0.1 seconds (rounded up), for example, 0.01 is reported as 0.1. The number of rows processed in each phase is provided by TPC-DI's data generator, DIGen.

We build our data generator using PDGF, the Parallel Data Generation Framework, originally developed at the University of Passau [30] ; since 2013, it has been commercialized by bankmark[1]. PDGF's core functionality, which includes parallel data generation of generic data types and complex inter- and intra-column data dependencies across multiple data, allowed us to implement a highly scalable data generator. With our extensions to PDGF, we can generate data for the initial load and subsequent incremental updates.

1.3.3 TPC-DS Version 2

To measure the performance of SQL-based big data systems, we utilize TPC-DS Version 1, a comprehensive benchmark for measuring the performance of DSS. We modified TPC-DS Version 2 in the areas of database load, ACID[2], incremental data integration, queries, metric and execution rules. Because big data systems follow an open data approach, in which all products in its ecosystem can access and modify the same full-fidelity data sets, we redefined what constitutes a database load and how the database system guarantees durability. We accomplished that by making the copying of data into a storage subsystem that is not exlusively owned by the data management system part of the benchmark load. Additionally, we dropped the ACI[3] requirements and require data read and write access during and after a permanent

[1]`http://www.bankmark.de`
[2]Atomicity, Consistency, Isolation and Durability
[3]Atomicity, Consistency and Isolation

irrecoverable failure of any single durable medium containing any database objects. Typical big data implementations of DSS do not necessarily undergo a rigid data integration process where dimension tables are updated; in TPC-DS V2 we focus on the loading of new and purging of old fact table data.

While most queries are carried over from TPC-DS V1 to TPC-DS V2, we modify some queries. Most of the modifications address inconsistencies between the functional query definition (SQL text) and its business description. Queries that returned non-deterministic query results were amended with order-by clauses. We also added equivalent rewrites to allow more big data products to run the queries. We conduct a thorough performance analysis of four systems using different technology, showing insights into relative query elapsed times within a system and across all four systems. We classify the queries using data scan analysis, coefficient of variation analysis and k-means analysis to show which queries are important and which may be dropped from the benchmark.

We redesigned the execution rules and metric in TPC-DS V2 to emphasize the performance characteristics of big data systems. To reflect the multi-user emphasis of big data systems, we require one single-user and two multi-user tests. The multi-user tests, in addition to executing queries, also execute the data integration workload.

Despite the pros and cons of using geometric means of calculating a single number to represent performance [31], we decided to change TPC-DS performance metric in Version 2, $QphDS@SF$, from an arithmetic mean to a geometric mean of the elapsed time for the four execution phases: (i) *Load*, (ii) *single-user*, (iii) *multi-user*, and (iv) *data integration*. This change addresses concerns by some TPC member companies that the original metric could, for some implementations, be dominated by data maintenance and load.

1.4 Contribution

The main contributions for the Internet of Things benchmark, TPC-IoT, are:

i. We develop TPCx-IoT, the first industry standard benchmark to measure the performance of IoT gateway systems.

ii. We provide an in-depth analysis of TPC-IoT with emphasis on its key features, and design decision, that is, we explain TPCx-IoT's underlying use case, how it is mapped to the benchmark specifications and the system being measured. We explain the execution rules, the ingest and analytical workloads and how the metric reveals the main performance characteristics of a typical gateway deployment.

iii. We explain how the technical ideas of TPCx-IoT are embedded in the TPC infrastructure that makes TPCx-IoT a robust industry standard benchmark. This includes pricing, auditing and reporting of TPCx-IoT benchmark results.

iv. We analyze how TPCx-IoT measures the performance of IoT gateway systems by presenting the performance measurements of TPCx-IoT against three different hardware configurations.

The main contributions for the Data Integration benchmark, TPC-DI, are:

i. We design TPC-DI, the first industry standard benchmark for measuring the performance of systems executing data integration processes for analytical analysis in data warehouses.

ii. We describe the benchmark in a deep and comprehensive way, discuss alternative designs wherever possible and provide reasons behind design decisions.

iii. We analyze the workload and present results obtained from a test system.

The major contributions of our TPC-DS approach to measuring performance of SQL-based big data systems are:

i. We designed the first industry standard benchmark for measuring the performance of SQL-based big data systems, TPC-DS V2. Based on TPC-DS V1, the development of TPC-DS V2 allows SQL-based big data platforms to complete a fully audited TPC-DS benchmark.

ii. We provide an in-depth analysis of TPC-DS V2 with emphasis on its key features, design decisions and workload differences to TPC-DS V1.

iii. We conduct experiments running all queries in single-user and multi-user modes on four analytical platforms, that resemble the diversity of systems being deployed in big data solutions: two big data systems, one traditional relational database system (RDBMS) and one columnar in-memory database system.

iv. We develop methods to categorize queries into equivalence classes, which give indicators for query redundancies and query deficiencies and apply these methods to the elapsed times gathered during our our experiments.

v. We analyze system resources consumptions during the execution of queries in single-user and multi-user runs.

Parts of the content and contributions of this work have been published in:

- M. Poess, T. Rabl, and H.-A. Jacobsen. "Analysis of TPC-DS: the first standard benchmark for SQL-based big data systems." In: *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24 - 27, 2017.* ACM, 2017, pp. 573–585. ISBN: 978-1-4503-5028-0. DOI: `10.1145/3127479.3128603`. URL: `http://doi.acm.org/10.1145/3127479.3128603` [32]

- M. Poess, T. Rabl, H.-A. Jacobsen, and B. Caufield. "TPC-DI: The First Industry Benchmark for Data Integration." In: *PVLDB* 7.13 (2014), pp. 1367–1378. URL: `http://www.vldb.org/pvldb/vol7/p1367-poess.pdf` [33]

- M. Poess, R. Othayoth, C. Narasimhadevara, K. Kulkarni, T. Rabl, and H.-A. Jacobsen. "Analysis of TPCx-IoT: The First Industry Standard Benchmark for IoT Gateway Systems." In: *Proceedings of the 34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-20, 2018.* [34]

1.5 Organization

The remainder of this document has been organized as follows. Chapter 2 provides a background on benchmarks in general and benchmarks from the TPC specifically. Chapter 3 presents the related work on the area of benchmarks for the DSS domain. Chapters 4 describes TPCx-IoT beginning with a description of its use case and continuing with a detailed description of the benchmark specifics. Chapter 4 concludes with a performance analysis of TPCx-Iot running on three hardware configurations. Chapter 5 presents details about the design of TPC-DI. After describing its use case, it dives into a detailed description of the underlying data model, both source and target. It continues with an in-depth description of the its data generation, PDGF and the data it generates. Before discussing the execution rules and metric, it explains how the transformations stress the capabilities of a data integration system. Chapter 5 concludes with a performance study. Chapter 6 covers TPC-DS. It is divided into two sections. The first section presents and analyzes TPC-DS, emphasizing the differences between Version 1 and Version 2 of TPC-DS The second section presents a detailed performance analysis of TPC-DS Version 2 on four different setups. Using cluster techniques the performance section analyzes the TPC-DS queries and proposes which queries do not contribute to the quality of the TPC-DS and which do. It concludes with recommendations about which types of queries to drop and which type of queries the TPC should develop more.

Chapter 2

Background

A computer benchmark is a methodology for quantitatively measuring the performance of a subset of a computer system, both hardware and software. At the minimum, a computer benchmark includes a workload and metric. The workload defines the work that the tested computer system must complete. The metric defines how performance is derived from the execution of the workload. Developing a benchmark specifically for a particular computer system is costly and time consuming. Such benchmarks have limited scopes, that is, they are designed for measuring the absolute performance of a very small subset of a computer system, typically during the design and development phases of new system features. Benchmarks designed for comparing the relative performance of multiple computer systems typically have larger scopes.

Independent of their scopes, benchmarks can be designed to be generic or domain specific. Generic benchmarks measure the performance of computer systems without considering the context in which they are used. Domain specific benchmarks are designed to measure the performance of computer systems running a particular family of applications, for example, database management software or scientific applications.

Benchmarks intended for publicly comparing the performance of computer systems are considered industry standard benchmarks. Industry standard benchmarks are developed by an industry standard benchmark consortia. They enable all system vendors to compete on a level playing field by providing fair performance comparisons. In addition to workload and metric, industry standard benchmarks must include rules to ensure fairness: (i) *disclosure requirements*, (ii) *benchmark compliance*, (iii) *realistic system design*, and (iv) *fair pricing* (if system price is part of the metric).

Disclosure requirements ensure that the reported performance is for a specific system configuration and that, in case changes were made to the benchmark, these changes remain disclosed. Benchmark compliance rules ensure that benchmark results are compliant with their benchmark standards. Rules for realistic system design ensure that the system being measured is a typical system a customer would deploy to run in the domain of the benchmark. For instance, an unrealistic system is a system that was specifically designed to run the benchmark and may not run other typical operations of the benchmark domain. Fair pricing rules are intended to ensure that the system being measured is priced in a realistic and fair way. For instance, an unrealistic or unfair pricing may include discounts not available for customers.

This work focuses on domain specific industry standard benchmarks.

2.1 Industry Standard Benchmark Consortia

Industry standard benchmark consortia are organizations that design, develop, maintain and publish industry standard benchmarks. They are mostly nonprofit organizations targeting a specific class of computer systems and class of domains. Well-known and respected industry standard benchmark consortia include the Standard Performance Evaluation Corporation (SPEC) [1], Storage Performance Council (SPC) [2], Embedded Microprocessor Benchmark Consortium (EMBC) [3] and the Transaction Processing Performance Council (TPC) [4]. They continue to develop benchmarks.

This work focuses on industry standard benchmarks for the TPC.

2.2 Transaction Processing Performance Council (TPC)

Under the leadership of Omri Serlin, the TPC was founded in 1988 as a nonprofit organization. The initial set of eight members— Control Data Corporation, Digital Equipment Corporation, Pyramid Technology, Stratus Computer, Sybase, Tandem Computers, and Wang Laboratories— grew to 35, a year later [9]. The mission of the TPC is "to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry."[35]. Following this mission, the TPC has provided the industry with 16 benchmarks in many domains, such as, Online Transaction Processing (OLTP), Decision Support (DS), Web Application (APP), Virtualization, Big Data and Internet of Things. These benchmarks produced over 1000 benchmark publications.

2.2.1 Organizational Structure

The main organ of the TPC is the *full council* (FC). Each member company of the TPC may send at least one representative to the meetings of the FC, which take place at least five times per year. Each company names one of its representative to be a director of the board. Each director is entitled to one vote. The FC is lead by one elected TPC director, the chair of the TPC. All major decisions are taken in the FC. Actual work is conducted in three types of committees, who meet regularly between FC meetings: (i) *Standing committees* and (ii) *Technical committees.*

There are three standing committees: (i) *Steering committee (SC)*, (ii) *Technical advisory board (TAB)*, and (iii) *Public relations committee.* The SC consists of five representatives from five different member companies and is lead by the chair of the TPC. The SC provides an overall direction of the TPC and oversees all administrative business of the TPC. The TAB is an arbitrator in settling disputes among member companies over benchmark interpretations and published benchmark results. The PR promotes the TPC, establishes TPC benchmarks as industry standards and organizes the TPCTC, TPC's annual technology conference.

Benchmarks are developed and maintained in technical committees lead by elected chairs. Each chair reports the work and decisions made in his subcommittee directly to the FC. There is one technical committee for each active benchmark standard: (i) TPC-C, (ii) TPC-DI, (iii) TPC-DS, (iv) TPC-E, (v) TPC-H, (vi) TPC-VMS, (vii) TPCx-BB, (viii) TPCx-HS, and (ix) TPCx-V, and each supporting benchmark standard: (x) TPC-Pricing and (xi) TPC-Energy. All chair positions and members of standing committees are elected annually for a one year term. The *Bylaws* [36] and *Policies* [37] of the TPC define and rule the operation of the TPC as an organization.

2.2.2 Benchmark Development in the TPC

The TPC provides mechanisms that facilitate the processes of turning a benchmark into an industry standard benchmark and maintaining it. These mechanisms are implemented in three locations: (i) *TPC policies* [37], (ii) *supporting benchmarks standards*, and (iii) *the benchmark specification.* The following sections describe these mechanisms in more detail.

2.2.3 Benchmark Result Certification

To ensure that the result of a benchmark execution is compliant with the spirit and letter of its benchmark standard, it must be certified by either a *TPC certified auditor* or a *pre-publication board* before its results can be published on the TPC website. A TPC certified auditor is an independent person, free of conflicts-of-interest who has successfully passed the TPC's auditor certification process. The certification process is done for each benchmark standard for which the auditor desires to be certified. Auditors who have passed the certification process for a particular benchmark standard have sufficient knowledge of it to certify benchmark results for it. The pre-publication board consists of one or more representatives from TPC member companies that have sufficient knowledge to certify a TPC benchmark result. Whether a benchmark standard requires a TPC certified auditor or pre-publication board is defined in the benchmark standard.

2.3 TPC Benchmark Classes

A *benchmark class* in TPC is a set of benchmark standards that share the same characteristics and the same rules for creation, maintenance, and publication. The TPC currently has two benchmark classes, namely, *enterprise* and *express*. Both approaches have merit. It depends on the domain of the benchmark whether it should be implemented as enterprise or express.

Enterprise class benchmarks, namely, TPC-C, TPC-E, TPC-H, TPC-DS, and TPC-DI, are defined in a technology agnostic way in the form of paper specifications. This paper specification defines a set of functional requirements that can be run on any system, regardless of specific hardware or software, as long as the system can fulfill the functional requirements of the specification.

Enterprise class benchmarks enable the performance evaluation of a broad spectrum of hardware and software tools, which is to be conducted in a fair and open way. They allow the use of technology that had not been developed when the benchmark specification was written. For example, when TPC-D was originally developed in the beginning of the 90's, no commercial product existed with aggregate/summary structures, that is, pre-computed auxiliary structures to speed up query execution. However, these are now common in many commercial database products.

On the contrary, enterprise class benchmark specification are complex. All functional requirements, such as workload, timings and durability characteristics need to be expressed in a hardware and software neutral way. Also, being technology agnostic renders it almost impossible to provide a kit that can be downloaded and run unmodified. Hence, each vendor using a specific hardware/software solution needs to develop their own implementation of an express class benchmark specification. This is expensive and time consuming as vendors have to submit proof that their implementations meet all benchmark requirements. This is usually done through an auditor.

Enterprise class benchmarks have served the TPC well for many years because when it was founded in 1988 there existed a myriad of database management software and computer vendors, some provided only database management software, such as Microsoft, some only hardware, such as Compaq and some provided both such as IBM. Through acquisition, consolidation and other market forces, the number of independent software and hardware vendors shrunk. At the same time, we saw a shift toward the construction of enterprise systems based on commodity hardware, especially, x86-based systems and open-source software. This convergence into fewer architectures has weakened the need for technology-agnostic specifications.

This led the TPC to develop the express benchmark framework. Express benchmarks are based on predefined executable software kits. They are restricted to the technology implemented in the kit. Consequently, they can be deployed rapidly with minimal modifications, while satisfying the rigid rules the TPC benchmarks are known for. There are many pros and cons of choosing a benchmark class for a given benchmark. For a more detailed discussion, see [38]. TPC currently supports four express benchmarks, TPCx-BB, TPCx-V, TPCx-HS, and TPCx-IoT.

Regardless of class, all benchmarks developed by the TPC share the same values of the TPC, namely:

- Publish only one performance metric
- Publish a price/performance metric
- Ensure availability of the measured system
- Ensure repeatability of a benchmark run
- Mandate a rigid audit process

The TPC breaks an enterprise benchmark standard into the following components, which it refers to as *clauses*:

- *Preamble*: In addition to introducing the benchmark on a very high, non-technical level, the *preamble* contains a clause that assures only commercially available products may be used for performance evaluation. This is important to prevent the use of “benchmark special” implementations that were designed only for the purpose of publishing a TPC benchmark result.

- *Business and benchmark model*: The *business and benchmark model* clause outlines the use-case of the benchmark. While each TPC benchmark may be applied to any industry that operates their system in the domain of the benchmark, each benchmark workload has been granted a realistic context. For example, TPC-DS models the decision support operations of a typical retail product supplier. The benchmark model aids the reader of the benchmark specification in relating intuitively to the benchmark.

- *Logical database design*: The *logical database design* clause describes the design of the database by abstraction. Column data types do not correspond to any specific SQL-standard data type. Instead these abstract data types define the minimum requirements for the properties of columns. A benchmark implementation may use any internal representation or SQL data type that meets the minimal requirements.

- *Scaling and database population*: The *scaling and database population* clause defines the minimal and maximal sizes of tables and how the tables are populated using the TPC provided data generator.

- *Workload overview*: The workload section defines the workload. For instance, in the case of TPC-C, this section defines the transactions while in the case of TPC-H it defines the queries and update functions.

- *ACID or data accessibility properties*: Keeping data accessible amid system failures is important for TPC benchmarks. While most TPC benchmarks do not measure the performance of system failures, all benchmarks require data to be accessible after a system failure.

- *SUT and driver implementation*: A System Under Test (SUT) is a term that describes all components of the system being tested. It contains priced and non-priced components. Each TPC benchmark defines the mandatory and optional parts of its SUT.

- *Full disclosure*: Defines what needs to be disclosed and how to organize the disclosure reports.

- *Audit*: Defines the minimum requirements for the audit process. The auditor is not bound by the content in this clause. On the contrary, an auditor must certify that the benchmark result is compliant with the entire benchmark specification.

2.4 Metric and Execution Rules

Execution rules and metric are two fundamental components of any benchmark definition and they are probably the most controversial when trying to reach an agreement between different vendors. The execution rules define the way individual pieces of a benchmark are executed and timed, while the metric emphasizes them by specifying their weight in the final metric. The metrics and execution rules are described in one section, as they are intrinsically connected to each other and they are equally powerful in controlling performance measurements. Both can change the focus of a benchmark because only those parts of a system that are executed, as described in the execution rules, can be measured in the metric. Conversely, even though a part is executed, if it is not timed and included in the metric, it will remain unnoticed. For instance, TPC-H's execution rules mandate the measurement of the initial database load. However, the primary metric (QphH) does not take the load time into account. Consequently, little consideration is given to it while running the benchmark.

TPC is best known for providing robust, simple and verifiable performance data. The most visible part of the performance data is the performance metric and the rules that lead to it. Producing benchmark results can be expensive and time consuming. Hence, the TPC's goal is to provide a robust performance metric, which allows for system performance comparisons for an extended period and, thereby, preserving benchmark investments. A performance metric needs to be simple such that easy system comparisons are possible. If there are multiple performance metrics, for example, A, B, C, system comparisons will become difficult because vendors can claim they perform well on some of the metrics, for example, A and C. This might still be acceptable if all components are equally important, but without this determination, it would lead to much debate on this issue. In order to unambiguously rank results, the TPC benchmarks focus on a single primary performance metric that encompass all aspects of a system's performance weighting the individual components. Taking the example from above, the performance metric M is calculated as a function of the three components A,B and C , for example M=f(A,B,C). Consequently, the TPC's performance metrics measure the system and overall workload performance rather than individual component performance. In addition to the performance metric, the TPC also includes other metrics, such as price-performance metrics.

2.4.1 Priced Configuration

The TPC is one of the few performance consortia that defines very strict rules for pricing a system and defining the availability of components of a system. Karl Huppler from IBM has summarized the pricing methodology and outlined the future direction of TPC pricing in [39].

Rules for pricing the *priced system* and its associated software and maintenance are included in the TPC *pricing specification* [40], which is listed under common benchmark specifications. Common benchmark specifications apply to all TPC benchmarks.

The system to be priced must include all hardware and software components listed in the SUT, a communication interface that can support user interface devices, additional operational components configured on the test system, and the maintenance of all above components. The cost of a priced system consists of the following costs:

- SUT price
- Price of additional products (software or hardware) in system
- Price of additional products (software or hardware) required for customary operation, administration and maintenance of the SUT for three years
- Price of all products required to create, execute, administer and maintain the executables necessary to create and populate the test environment

The following components are excluded from the cost of a priced system:

- End-user communication devices including related cables, connectors, and switches
- Equipment and tools used exclusively for FDR[1] production

Pricing is strictly associated with the availability of system components. If any component of a benchmark becomes unavailable, the benchmark result must be withdrawn. If, however, components with comparable performance are available, pricing rules allow for component substitutions. If corrections to components of the priced configuration are required during the life of a product, these changes are not considered substitutions as long as the part number of the priced component is identical to the new component. The idea behind this change is that hardware and software suppliers may update the components of the priced configuration so long as these updates do not negatively impact the reported performance metric or numerical quantities by more than two percent.

The following are not considered substitutions:

- Software patches to resolve a security vulnerability
- Silicon revision to correct errors
- New supplier of functionally equivalent components, for example, memory chips and disk drives
- Durable Media (for example, disk drives) and cables[2]

[1]Full Disclosure Report

[2]A durable medium is defined as a data storage medium that is inherently non-volatile, such as a magnetic disk or tape

2.4.2 Full Disclosure Report and Executive Summary

Each benchmark results is required to provide a *full disclosure report* (FDR) and *executive summary* (ES). The intent of these disclosures is to simplify comparisons between results and for facilitating replication of the results of any given benchmark, given appropriate documentation and products. The FDR must contain all customer-tunable parameters and options that have been changed from the defaults found in actual products, including, but not limited to, the configuration parameters and options of the operating system, server, storage, network, and any other hardware components incorporated into the pricing structure. In addition if any software is specifically compiled for running a benchmark, compiler optimization options, if any, must be disclosed.

Furthermore, the FDR must list diagrams of both the measured and the priced configurations, along with a description of the differences. The diagrams must clearly show the number of nodes used and the total number and types of processors used including sizes of L2 and L3 caches. The diagrams must further show the size of memory; any specific mapping/partitioning of memory that is unique to the SUT; number and type of disk units (and controllers, if applicable), number of channels or bus connections to disk units, including protocol type, number and speed of LAN connections and switches; and any other hardware components physically used in the test or incorporated into the pricing structure.

Chapter 3

Related Work

The need for methodologies to measure the performance of computer systems dates back to the development of the first systems. The need for industry standard benchmarks for data processing systems, however, arose in the beginning of the 90's when companies began engaging in "benchmarketing". Instead of using the same methods for measuring the performance of their system, vendors defined benchmarks that highlighted the performance of their system. This made performance comparison virtually impossible. One of the first publications of industry standard benchmarks is Jim Gay's "The Benchmark Handbook: For Database and Transaction Processing Systems" [41]. In his introduction, Jim Gray motivated the need for *domain specific* benchmarks [42]. Most modern industry standard benchmarks are domain specific.

This chapter is divided into three sections. The first section presents the related work in the area of Internet of Things. The second section presents the related work in the area of ETL and Data Integration, and the third section lists the related work in the area of back-end decision support systems.

3.1 Internet of Things

While there have been industry standard benchmarks for measuring the performance of IoT edge devices and datacenter analytic systems, TPCx-IoT is the first industry standard benchmark for measuring the performance of IoT gateway systems. As TPCx-IoT is an express benchmark that provides a working kit, the TPC specification is limited to providing high-level information. This thesis provides a detailed description of TPCx-IoT.

The nonprofit Embedded Microprocessor Benchmark Consortium (EMBC) has developed and is actively supporting *IoTMark* [43], a suite of micro-benchmarks for analyzing edge devices. The benchmark suite focuses on measuring the energy consumption of the three main parts of an edge node, *sensors*, *processing*, and *communication protocol*. IoTMark is based on real-world use cases and, therefore, determines the combined energy consumption of the entire edge platform (sensor interface, processor, and radio interface). However, none of the benchmarks published by the EEMBC include any performance or price-performance metric in contrast to TPCx-IoT.

IoTABench is an experimental benchmark toolkit [44]. It currently implements a smart meter use case (electricity) [45], which includes the loading of synthetic data, performing simple data cleansing, and six analytical queries. The synthetic data is generated by a Markov chain data generator that was trained using real data from Irish households. The data-cleansing step requires the insertion of missing data (1% dropped samples) and the six analytical queries perform *projection*, *aggregation*, *selection*, and *order by* operations. The sophisticated data generator of IoTBench generates very realistic data in parallel. IoTABench models a data ingestion rate of about 67,000 smart meter readings per second. In comparison, TPCx-IoT models millions of inserts per second. IoTABench models a single sensor type, while TPCx-IoT models 200 sensor types. IoTABench measures the data ingestion rate and query performance sequentially, while TPCx-IoT measures insert and query operations concurrently. The number of queries executed by the two benchmarks also differs. While TPCx-IoT executes five queries for every 10,000 sensor readings, IoTABench executes a total of 12 queries (two times six different queries). With this model, IoTABench can be considered to be a datacenter and not a gateway benchmark. In addition, IoTABench is not an industry standard benchmark.

TPC-H [14] is an industry standard benchmark that is used extensively in industry and academia for measuring the performance of data warehouse systems, which are deployed in IoT datacenter back-end systems. TPC-H requires an ACID compliant database system.

3.2 Data Integration

Little research has been conducted on benchmarks for ETL systems. Today, most systems are tested with rather simple workloads such as loading TPC-H data (cf., [46, 47, 48]). A more involved proposal was presented by Manapps [49]. In this benchmark, 11 independent ETL jobs were used for comparing 5 ETL systems. These jobs included simple loading, joining two tables, and aggregations. This is a typical example of a component benchmark, which singularizes certain features of the system under test. While this is beneficial for understanding the bottlenecks in a ETL workload, it does not necessary relate to real world performance, since the interplay of operations has a significant impact on the performance. Therefore, TPC-DI was developed as a full end-to-end benchmark with a complex workload, giving a realistic view of the end-to-end system performance.

Vassiliadis et al. characterized patterns in ETL workflows, as well as metrics and parameters a ETL benchmark should cover [50]. These were extended to a complete ETL workload based on TPC-H [51]. In contrast to TPC-DI, this benchmark contains only simple transformations that can be handled with regular SQL constructs and, therefore, do not necessarily stress elaborate ETL systems.

3.3 Back-End Decision Support

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 the analytic features of SQL-based database engines. While there have been two publications describing TPC-DS V1 [52, 53], they only describe the 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 thesis describes the second version of the TPC benchmark in its released form and touches on its differences with Version 1.

Several other benchmarks have been proposed for big data benchmarking. An overview of benchmarks can be found in [54]. In the following, we will only discuss the 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 [55, 56]) or benchmarks targeted only at a single type of system (MapReduce [57] or Spark [58]).

TPC-DS V2 is not the first big data benchmark to be 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 [59]. 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 [60, 61], 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 only has 30 queries, 10 of which are based on TPC-DS V1 queries. The remaining queries address use cases that are difficult to express in standard SQL. In general, BigBench targets read-only big data platforms 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 include Pavlo's benchmark [62], its successors HiBench [63], and the AMPLab Benchmark [64]. 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 for any kind of SQL engines as well.

Chapter 4

TPCx-IoT: First Industry Standard Benchmark for Measuring the Performance of IoT Gateway Systems

Without doubt, the IoT has been an influential key driver of innovation, both in the consumer and business segments of many industries. The initial hype around the IoT stemmed from typical consumer use cases such as wearable fitness trackers, smart watches, and smart home devices. However, the number of devices and data from enterprise use cases such as smart city, patient healthcare, preventative maintenance, and smart power grid have a significant market share. Regardless of use case, the IoT will continue to drive innovation of physical devices, networks, and back-end analytical infrastructure. According to a 2017 projection by Gartner [24], the total number of IoT devices will more than double between now and 2020 (8.4 Billion to 20.4 Billion). The breakdown between consumer and business devices will remain steady at 62% and 38%, respectively.

Figure 4.0.1 shows a simplified three-tier architecture of a typical IoT system with *edge devices* on the left, *gateways* in the middle, and *datacenter* on the right. Data

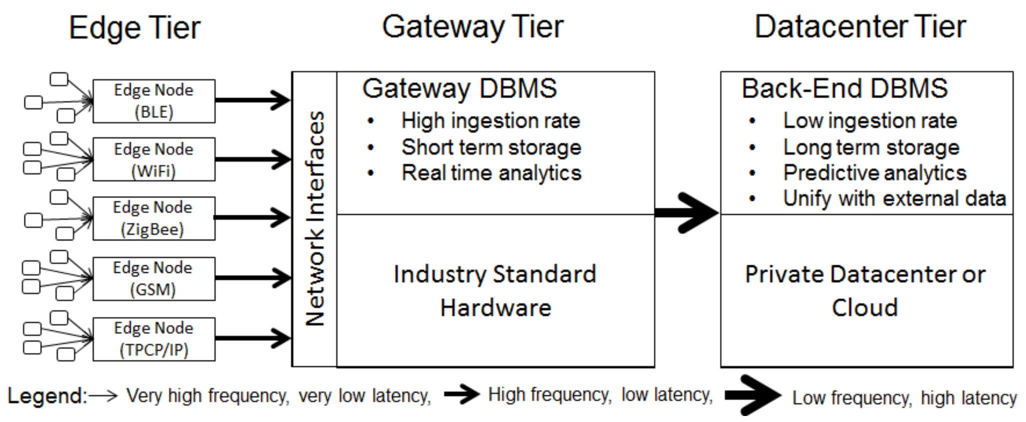


Figure 4.0.1: Schematic overview of a typical IoT infrastructure

flows with very high frequencies and very low latency requirements from a myriad of sensors into edge devices, where the typically analog signals are converted into digital data. The frequencies at which sensors generate data depend on the sensor type. For instance, a *phasor measurement unit* (PMU), which measures electrical waves using synchrophasors [65], can generate 60-121 readings per second. Vibration sensors for monitoring machine health as part of predictive machine maintenance are capable of generating data at a much higher rate of thousands of samples per second (sps). The edge devices then send that data at high frequencies with low latency requirements by using a variety of network protocols, such as TCP/IP, Bluetooth Low Energy (BLE) [66], ZigBee [67], Wireless Internet (WiFi) [68], Global System for Mobile (GSM) [69], and MQTT [70], to gateway systems. Acting as a single point of access for these devices, gateways perform the functions of short-term persistent storage and lightweight local analytics (filter, aggregation) in their own DBMS, which often runs open-source software on industry standard servers[1]. With these capabilities, gateways serve as a crucial monitoring tool for a selected area of an operational field by providing dashboard-like functionality. The back-end system on the right side of Figure 4.0.1 ingests data from, potentially, multiple gateways at low frequencies, for example, once a day, without much latency requirement. It stores data for a long term and performs complex global analytics.

Analytics that run on gateways are restricted to the data sent from the edges connected directly to a given gateway. Typical analytical queries in gateways run

[1]Usually x86-based servers built with commodity hardware parts.

frequently and include data aggregation, duplicate removal, and outlier and error detection. They usually operate on a subset of the data, for example, they are restricted to a specific sensor or date range (ingested within the previous minutes). As they are run very frequently and, potentially, concurrently with other queries, they are required to be completed in sub-second.

Analytics that run on datacenters usually span multiple gateways and consider data from previous weeks and years. They include both reporting and ad-hoc queries, which tend to be complex, long-running and based on a large subset of data. Usually, few queries run concurrently, and they are allowed to run for an extended period.

Foreseeing a massive increase in the volume of data originating from edge devices that need to be processed by gateway systems with high reliability and performance, the Transaction Processing Performance Council (TPC) [4] released TPC Express Benchmark IoT (TPCx-IoT) [71], the first industry standard benchmark for gateway systems. Its first version was released in May 2017. Following the tradition of previous TPC benchmarks, TPCx-IoT provides an objective measure of hardware, operating system, data storage, and data management systems to the industry and academia with verifiable performance, price-performance, and availability metrics.

TPCx-IoT specifically targets the gateway tier because the existing industry standard benchmarks already cover the edge and datacenter tiers. IoTMark, developed by Embedded Microprocessor Benchmark Consortium (EEMBC), is a suite of IoT connectivity benchmarks for testing and analyzing microcontrollers and connectivity interfaces of edge devices. The datacenter tier is covered by the TPC's own benchmarks, TPC-H and TPC-DS. Covering gateway systems seemed a natural fit for the TPC as it focuses on data intensive applications. TPC discussed combining benchmarks from all three tiers into one end-to-end benchmark, but it concluded that there would be too many different devices and software solutions distributed by different vendors in such an end-to-end benchmark. This would make benchmarking extremely time consuming and expensive. In addition, interpretation of the benchmark results would become extremely difficult.

TPCx-IoT provides an objective measure of hardware, operating system, data storage, and data management systems to offer the industry with verifiable performance, price-performance, and availability metrics for systems that are meant to ingest and persist massive amounts of data from a large number of devices and provide real-time insights, typical in IoT gateway systems running commercially available systems, both software and hardware. The TPCx-IoT benchmark models a continuous system that is available 24h a day, 7 days a week. It can be used for assessing a broad range of system topologies and implementation methodologies in a technically rigorous, directly comparable, and vendor-neutral manner.

The remainder of this chapter is divided into five sections. The first section describes TPCx-IoT in detail, including its use case, execution rules, metrics, data ingestion workload, query workload and workload driver. The second section explains how TPCx-IoT is embedded in TPC's benchmark infrastructure, including a description of its benchmark class, and its pricing and auditing requirements. In the third section, we demonstrate how TPCx-IoT measures the performance of real systems by analyzing TPCx-IoT performance data gathered from different hardware configurations running HBase.

4.1 Description of TPCx-IoT

This section provides a detailed description of the design of TPCx-IoT, including background information for key decisions. For the full specification and kit download of TPCx-IoT, please refer to the TPC website [6].

4.1.1 Use Case Description: Power Substations of Electric Utility Providers

While the performance data of TPCx-IoT may be applied to any IoT installation that must ingest and persist massive amounts of data from a larger number of sensors, and provide real-time analysis of incoming data, the workload of TPCx-IoT has been granted a realistic context. It models the power substations of a typical electric utility provider.

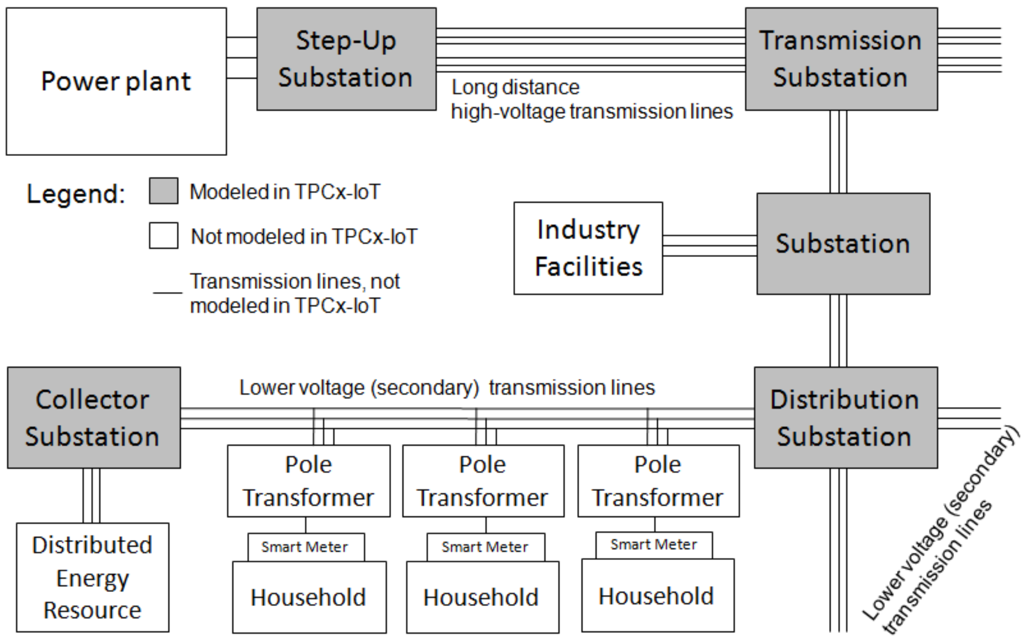


Figure 4.1.1: High level description of TPCx-IoT's use-case

A schematic description of an electric utility provider's infrastructure has been illustrated in Figure 4.1.1. It consists of *power producers*, for example, power plants and distributed energy resources (DERs), a *power grid*, and *power consumers*.

To transport power from producers to consumers, it needs to be converted to various voltages and distributed over a grid of transmission lines. Situated at every junction of this power grid are *power substations* that undertake a variety of tasks: *Step-up transmission substations* increase the voltage for the efficient movement of large amounts of electricity over long distances. *Step-down transmission substations* do the reverse, that is, reduce voltage as the electric power approaches its destination. *Transmission substations* connect two or more transmission lines. They contain high-voltage switches that allow transmission lines to be connected or disconnected for maintenance. *Distribution substations* transfer power from the transmission system to the distribution system of an area. In case power needs to be fed into the power grid from distributed power generation, for example, wind farms, *collector substation* are used. They allow power flow in the opposite direction, back into the transmission grid. According to a 2017 report of the California Energy Commission [72], the power grid of the state of California includes 3,200 power substations, of which 982

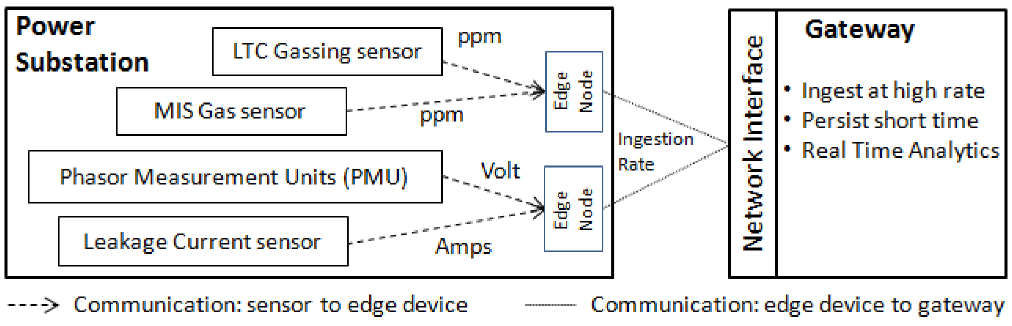


Figure 4.1.2: Overview of a gateway architecture serving power substations

are owned by one utility provider, Pacific Gas and Electric Company (PG&E) [73].

Utility providers use IoT technology in every step from thousands of edge sensors in power plants to a few sensors measuring power consumption at consumer sites. Requirements for gateways serving edge devices in each step vary dramatically. For instance, fossil fuel power plants deploy upto 4,000 sensors [74], while neighborhood smart meters only contain one. Hence, for TPCx-IoT, we decided to model sensor data generated by power substations with 200 sensors each. Power producers, consumers, and the actual transmission lines are outside of the scope of TPCx-IoT, as indicated by the non-shaded boxes in Figure 4.1.1.

The size of power substations vary depending on their purpose and the number of customers they serve. For instance, the Larkin power substation in downtown San Francisco [75] measures 12,200 square feet, while the Martin power substations [76] occupies 319,000 square feet. Power substations contain various types of sensors [77]. Figure 4.1.2 illustrates four examples: *load tap changers gassing sensors* measure gas levels in load tap changes (LTC) that regulate output voltages of transformers. Gas levels can identify overheating, coking and worn contacts of an LTC. *Metal–insulator–semiconductor (MIS) gas sensors* measure H_2 and C_2H_2 levels [78]. *Phasor measurement units* (PMU) measure electrical waves using synchrophasors [65]. *Leakage current sensors* measure the amount of current leakage to earth (ground). These sensors send their data via edge devices to a gateway that can either be situated close to the power substation or in a datacenter/cloud. The gateways ingest sensor data at a high rate, store that data for short durations, and provide real-time analytics to help take immediate action in associated power substations. Gateways

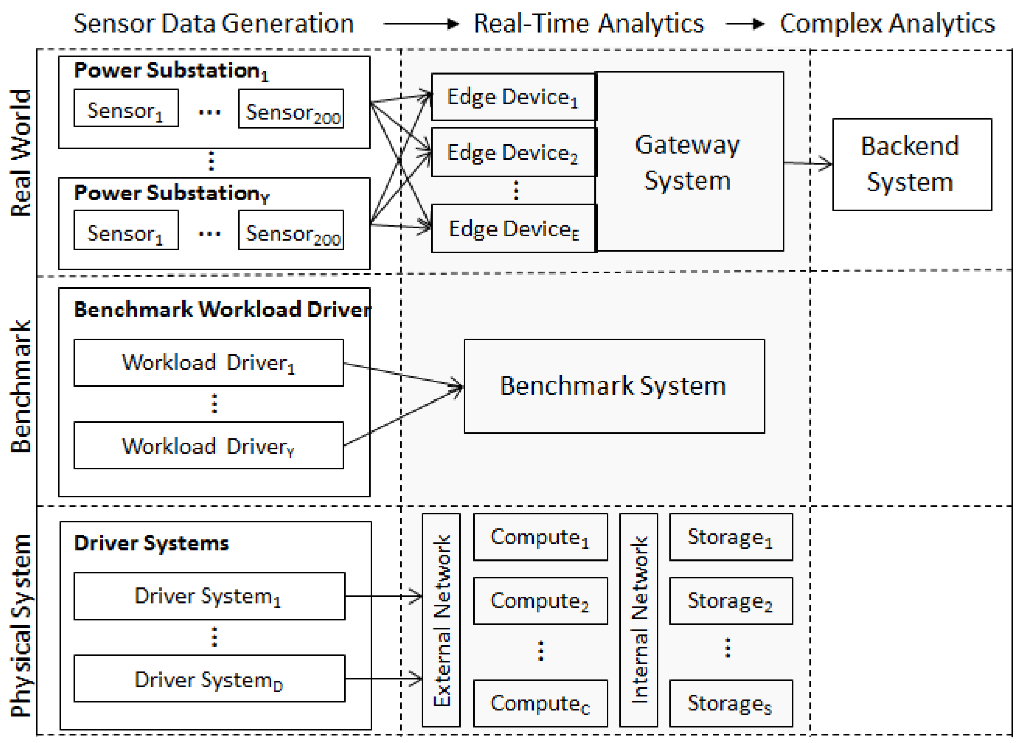


Figure 4.1.3: Mapping of real-world, benchmark, and physical devices

send data for long-term storage and complex analytics to an analytical platform. This is, however, out of the scope of TPCx-IoT.

Figure 4.1.3 shows how the real-world use case of power substations (see Figure 4.1.1) is mapped to the specification of TPCx-IoT and how it is mapped to physical systems. Each power substation in the simulated "real-world" maps directly to one instance of *TPCx-IoT workload driver* on the benchmark level. One instance of the workload driver generates sensor data from one power substation and the corresponding queries that can be run against that data. The physical level shown at the bottom of Figure 4.1.3 lists the number of physical systems needed to support the number of simulated power substations, that is, the number of driver instances that can be run. As long as the run-time requirements for the driver are being met, driver instances can be run on one or across multiple physical systems. These systems are not priced.

The gateway supporting our power substations is mapped directly to the SUT.

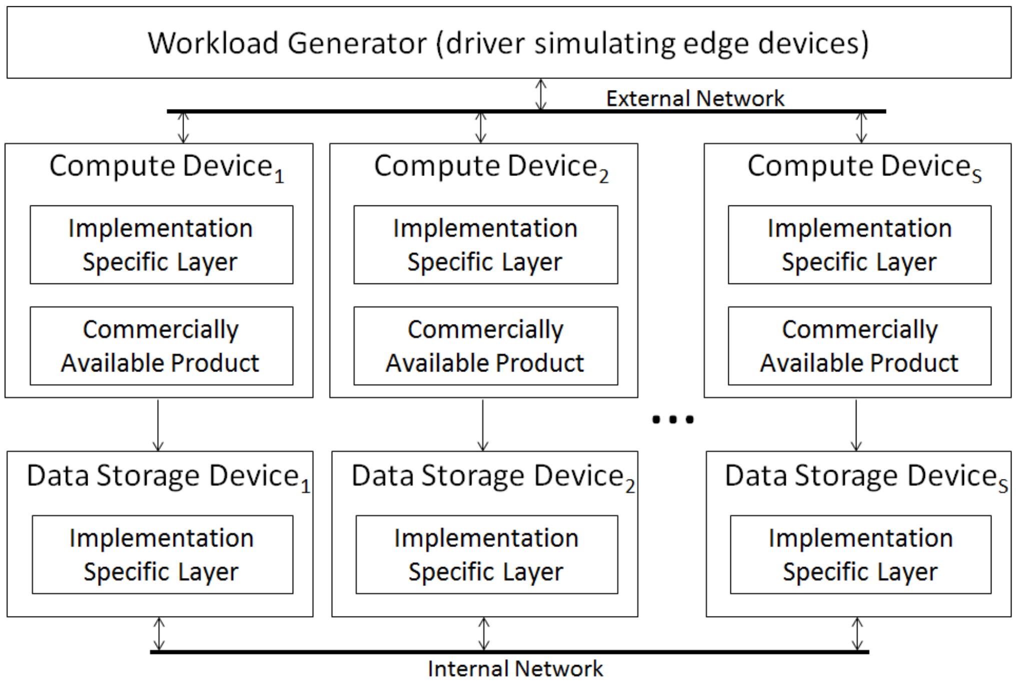


Figure 4.1.4: TPCx-IoT system under test (SUT)

TPCx-IoT defines the SUT as representing an IoT gateway system consisting of commercially available servers, switches, and storage systems running a database management system, for example, a NoSQL database. Figure 4.1.4 shows a diagram of the SUT as defined in the TPCx-IoT specification.

4.1.2 Execution Rules

A TPCx-IoT *benchmark run* is composed of two *benchmark iterations* (see Figure 4.1.5). Each benchmark iteration executes the TPCx-IoT *workload* twice, once to warm up the system and once for a measured execution of the workload. Before the first warm-up run, the benchmark driver performs a couple of prerequisite checks: The *file check* compares the checksums (md5sum) of all non-changeable kit files[2] on the system with reference checksums in the kit; The *data replication check* ensures

[2]Most files in the kit must not be altered by a benchmark sponsor. However configuration related files may be altered.

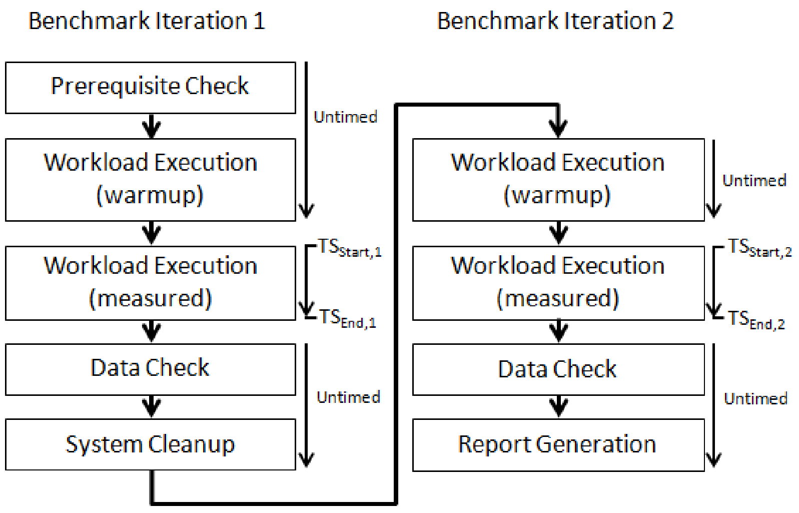


Figure 4.1.5: Benchmark execution rules

that the storage system uses a three-way replication of data. If any of these two checks fail, the benchmark driver will abort the run.

The warm-up run is not timed. The start and end timestamps of the measured run are denoted by $TS_{a,b}$ where $a \in \{start, end\}$ and $b \in \{1, 2\}$. The elapsed time of a workload execution is TE_a where $a \in \{1, 2\}$. Each *workload execution* performs concurrent data ingestion (write) and query (read) operations.

The amount of data to be ingested is a parameter of the workload driver. After the measured workload execution has been completed, a *data check* assures that the benchmark run fulfills all necessary runtime requirements. The second benchmark iteration is a repetition run to ensure measurement repeatability. To achieve identical conditions in both runs, a *system cleanup* is performed between iterations. A cleanup consists of purging all data ingested into the data management system during the previous warmup and measured workloads, deleting all temporary files, and restarting the data management system. No additional activities are allowed between the end of the first benchmark iteration and start of the second one. After the system cleanup has been completed, the second iteration is executed. After data check of the second iteration, the TPCx-IoT driver runs a report that prints all information needed to audit and publish a benchmark result.

For each benchmark run, the benchmark sponsor must choose the amount of ingest data and the number of simulated power substations to fulfill the following two execution rule requirements:

1. *Workload execution elapsed time*: Both the warmup and the measured workload execution need to run for at least $1800s$ each.
2. *Sensor data ingest rate*: The benchmark system must guarantee a minimal average data ingest rate per sensor of $20\frac{kvps}{s}$.

The acronym *kvp* stands for key-value-pair, plural is *kvps*. It is commonly used in NonSQL systems for describing a set of two data items, *key* and *value*. The key is a unique identifier for the value. In our use case, *kvp* stand for the sensor readings arriving to the gateway system from edge devices. In the remainder of this work, we shall use the terms $kvp(s)$ and sensor reading(s) interchangeably.

Each workload run must be at least $1,800s$ so that the benchmarked system can demonstrate that it can sustain high performance during an extended period before the data is forwarded to analytical systems in the back-end.

A minimal average data ingest rate per sensor is required to prevent benchmark sponsors[3] from artificially reducing the amount of reads required by queries. Assuming a fixed number of sensors, there is a direct correlation between the system-wide data ingest rate per sensor and the average number of readings retrieved by each query, because each query reads data from a $5s$ interval. Keeping the system-wide data ingest rate constant, because the gateway system is saturated, one could reduce the average per sensor ingest rate by increasing the number of sensors, that is, by increasing the number of power substations, that is, the number of TPCx-IoT driver instances.

Requiring a minimal average ingest rate of $20\frac{kvps}{s}$ per sensor has a couple of other consequences, which we will discuss below. As one power substation has 200 sensors,

[3]Vendors producing benchmark results and publishing them under TPC rules.

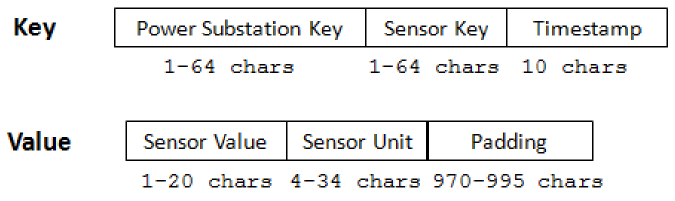


Figure 4.1.6: Sensor reading (*kvp*) generated by the driver program of TPCx-IoT

the system-wide minimal average ingest rate is:

$$200 sensors * 20 \frac{kvps}{sensors * s} = 4000 \frac{kvps}{s} = 3.91 \frac{MB}{s} \tag{4.1.1}$$

The minimal average number of *kvps* retrieved per query is:

$$20 \frac{kvps}{sensor * s} * 5s = 100 \frac{kvps}{sensor} \tag{4.1.2}$$

4.1.3 Data Ingestion Workload

The TPCx-IoT workload generator is based on the Yahoo! Cloud Serving Benchmark framework (YCSB) [28]. We chose YCSB for developing of TPCx-IoT because it is well-known, open source, easily adaptable to specific needs, and its built-in database interface layer allows connections to many common open-source database management systems including NoSQL DBMS.

Before diving into the details of the changes, we briefly review the requirements of the data ingestion part of our workload driver. As outlined in the use case description (Figure 4.1.3) the workload driver of TPCx-IoT must be able to generate sensor data related to different power substations. In real life, power substations vary in size depending on the number of transmission lines they connect or the number of customers they serve. The number of sensors in a substation varies accordingly. However, in our model, we assume that each substation has the same number of sensors, that is, 200.

To distinguish sensor data from different power substations, we added support in

YCSB for keys and values that are based on attributes from sensors commonly deployed in power substations of utility companies. Each YCSB instance, which we refer to as a TPCx-IoT driver instance, generates sensor data arriving from one power substation. The power substation key is passed to the TPCx-IoT driver instance along with the number of sensor readings it should generate (SR). Figure 4.1.6 shows the structure of key-value pairs. Each represents one reading of one sensor of one power substation. The *key* part consists of the *power substation key*, which uniquely identifies a power substation, the *sensor key*, which uniquely identifies a sensor within a power substation and the *timestamp*, coded as *POSIX time*, which represents the time the sensor reading was taken. The *value* part consists of the *sensor value*, which represents the value that the sensor read, *sensor unit*, which represents the measurement unit of the sensor value, and *padding*, which contains random text to fill a *kvp* to one KByte.

Figure 4.1.7 illustrates the bare generation speed at which TPCx-IoT drivers generate *kvps*. We measure this speed by redirecting the driver's output to */dev/null*. These experiments were conducted on a Cisco UCS C220 M4 driver system with 128GB main memory and two Intel Xeon 2680 v4 CPUs running at 2.4 GHz. Each socket had 14 cores, 28 multi-threaded. On the x-axis, we varied the number of TPCx-IoT drivers from 1 to 64. The left y-axis shows the aggregated driver throughput in thousand *kvps* per second ($\frac{kvps}{s}$). The right y-axis shows the CPU utilization of the driver system in percentage. As we increased the number of TPCx-IoT drivers from 1 to 64, the total throughput increased from 120,000$\frac{kvps}{s}$ with one driver to 1.1 Million$\frac{kvps}{s}$ with 32 drivers, while CPU utilization increased from 4% with one driver to 75% with 32 drivers. Total throughput drops to 900,000$\frac{kvps}{s}$ with 64 drivers with CPU utilization increasing to 100% with the system CPU portion increasing from 5% to 15%. This is not surprising because 64 drivers spawn 640 threads, which impose overheads of garbage collection and scheduling. The above generation speed of 1.1 million$\frac{kvps}{s}$ is sufficient to simulate the amount of data generated by one power substation. Each TPCx-IoT driver simulates the data volume of one power substation with 200 sensors. That means, 1.1 million$\frac{kvps}{s}$ generated with 32 drivers simulates a sensor frequency of about 170 samples per second (sps) per sensor.

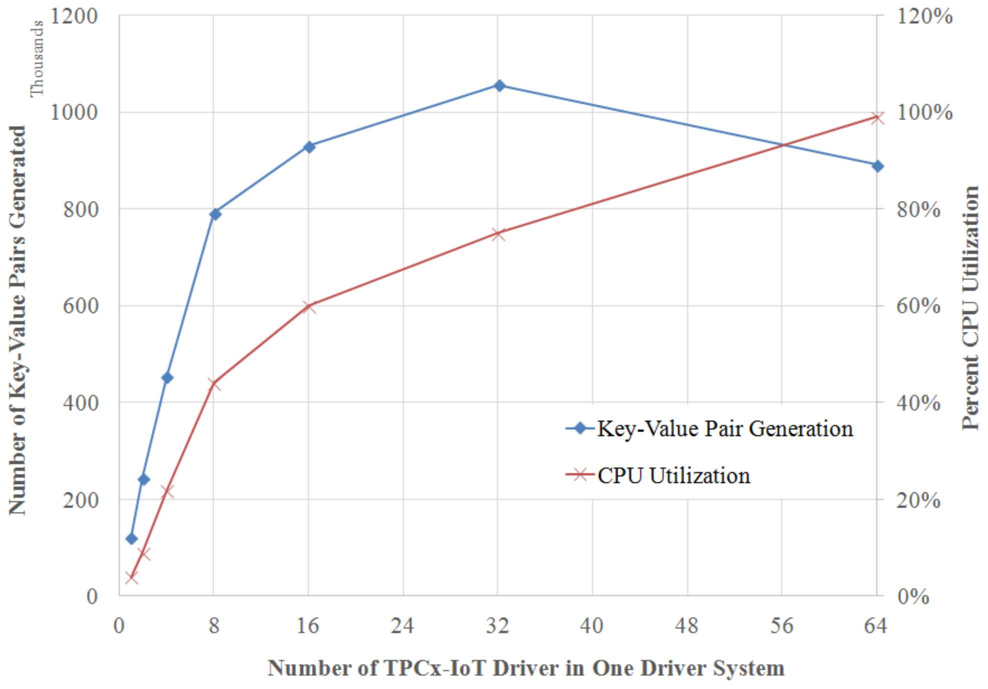


Figure 4.1.7: Key-Value pair generation speed

4.1.4 Query Generation/Execution

We added support for the concurrent querying of ingested sensor data. Unlike in the traditional YCSB deployments, where YCSB picks a random set of keys to read, TPCx-IoT issues queries that read random key ranges. Queries are generated from the following four query templates, which represent typical dash-board-like queries:

1. *Max-Reading*: The max-reading query compares the maximum sensor reading in the two intervals.
2. *Min-Reading*: The min-reading query compares the minimum sensor reading in the two intervals.
3. *Average-Reading*: The average-reading query compares the average sensor reading in the two intervals.
4. *Reading-Count*: The sample-count query compares the number of sensor readings in the two intervals.

Each query compares the readings of one sensor of one power substation ingested in the last $5s$ with data from another $5s$ interval that is randomly selected within the previous $1800s$. We chose the interval of the second query to be randomly picked, as opposed to picking it from a fixed time window to minimize caching effects. All four query templates perform projections, selections, and aggregations. The projection returns the fields required by the query, namely sensor value and time stamp. The selection filters data relevant to a specific power substation, sensor, and date range. The aggregation performs max, min, average, and count operations. Listing 4.1.1 shows a sample query.

As the database is empty before the start of the warm-up run, queries issued during the warm-up run might not return any data for the second, randomly chosen time interval, because there might be an absence of data in that time interval. This is not a problem as the warm-up run is not timed nor is any information from it part of the metric.

```
Scan s=new Scan();
ResultScanner scanner = null;
try {
 s.setTimeRange(timestamp,timestamp+ 5000);
 StringBuffer startKey=new StringBuffer();
 startKey.append(clientFilter);
 startKey.append(":");
 startKey.append(filter);
 startKey.append(":");
 startKey.append(timestamp);
 StringBuffer endKey=new StringBuffer();
 endKey.append(clientFilter);
 endKey.append(":");
 endKey.append(filter);
 endKey.append(":");
 endKey.append(timestamp+5000);
 s.setStartRow(startKey.toString().getBytes());
 s.setStopRow(endKey.toString().getBytes());
 if (fields==null) {
   s.addFamily(columnFamilyBytes);
 } else {
 for (String field:fields) {
   s.addColumn(columnFamilyBytes,Bytes.toBytes(field));
 }
}
scanner = currentTable.getScanner(s);
int numResults = 0;
for (Result rr = scanner.next();
       rr != null;
       rr = scanner.next()) {
  String key = Bytes.toString(rr.getRow());
  if (debug) {
    System.out.println("Got result for key: " + key);
  }
 }
}
```

Listing 4.1.1: Sample Query Code

4.1.5 Benchmark Driver

Figure 4.1.8 shows the architecture of the benchmark driver of TPCx-IoT. It is responsible for running the entire workload: performing prerequisite checks, performing data inserts (write operations), executing queries (read operations), checking data, performing system cleanup, and, generating reports. It serves as a wrapper around the TPCx-IoT driver instances, which are based on YCSB. Many of the TPCx-IoT driver instances are spawn depending on the gateway size.

The benchmark driver is invoked with two arguments, *number of TPCx-IoT driver instances* and *total number of kvps*. The number of TPCx-IoT driver instances determines the number of processes that generate data, the number of power substations that are simulated and the number of different sensors that send data to the gateway. The total number of *kvps*, which simulate the number of sensor readings sent to the gateway, determines how many *kvps* are ingested into the gateway. The default number of *kvps* is 1 Billion. By increasing the number of *kvps*, the benchmark sponsor can adjust the run time of the workload run. As the metric is throughput-based, longer runs offer no advantages.

Each TPCx-IoT driver instances i generates about the same number of *kvps*, $KVP(i)$. With P being the number of simulated power substations and K being the total number of *kvps* generated by all drivers, the number of *kvps* generated by each driver can be calculated as follows:

$$KVP(i) = \begin{cases} \lfloor \frac{K}{P} \rfloor & \text{if } 1 \leq i < P \\ \lfloor \frac{K}{P} \rfloor + K \ mod \ P & \text{otherwise} \end{cases} \tag{4.1.3}$$

As real benchmark configurations are expected to use values for K that are much larger (Billions) than P (hundreds), the above equation will not introduce a significant workload skew.

4.1.6 Metrics

TPCx-IoT defines three primary metrics:

1. Performance metric: IoTps
2. Price-performance metric: \$/IoTps and
3. System availability: Date

The performance metric reflects the effective gateway ingestion rate in seconds that the SUT can support during a 30-minute measurement interval, while executing

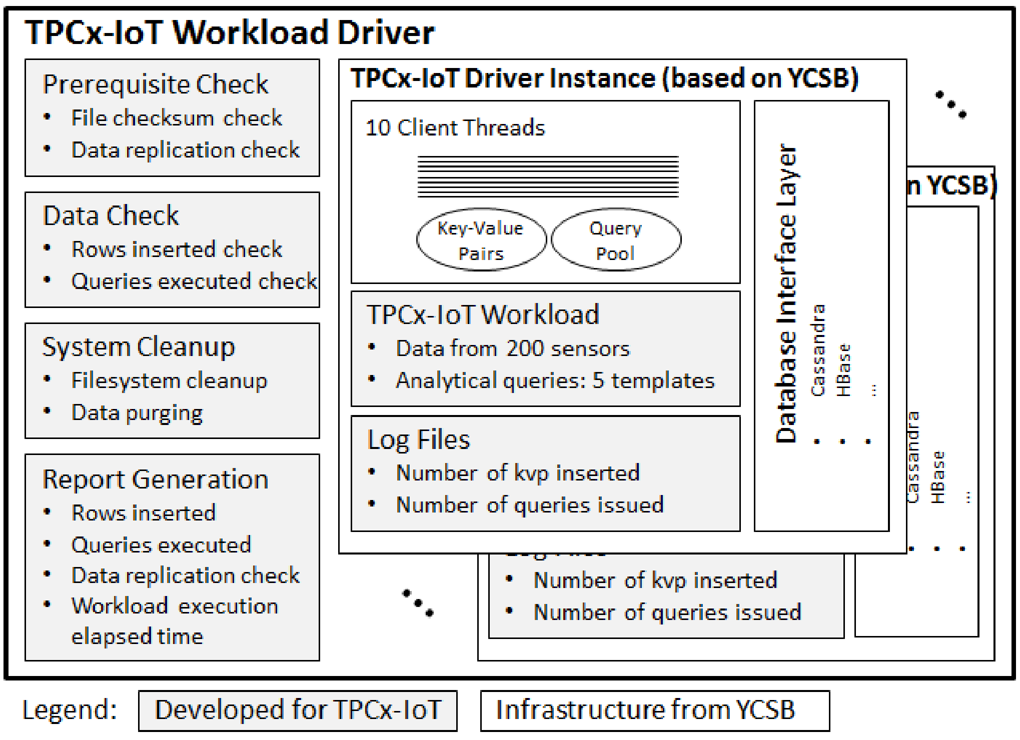


Figure 4.1.8: Architecture of TPCx-IoT workload generator

real-time analytic queries. The total number of *kvps* ingested into the database, N_i, where $i \in \{1, 2\}$ being the iteration, is used to calculate the performance metric. The performance run is defined as the measured run m with the lower number of *kvps* ingested, that is, $m \in \{1, 2\}, n \in \{1, 2\}$ such that $m \neq n$ and $N_m < N_n$. IoTps is calculated as follows:

$$IoTps = \frac{N_m}{TS_{end,m} - TS_{start,m}} \tag{4.1.4}$$

The price-performance metric reflects the total cost of ownership per unit IoTps performance. It is calculated as follows:

$$\frac{\$}{IoTps} = \frac{ownership\ cost * (TS_{end,m} - TS_{start,m})}{N_m} \tag{4.1.5}$$

The system availability metric reflects the date when all line items of the price configuration are generally available,that is, to any costumer. The system availability metric is important because it guarantees that the benchmarked system is a production system that can be purchased and not an experimental system that was implemented only for running the benchmark.

4.1.7 Express Model for TPCx-IoT

We selected the express model for TPCx-IoT, because the gateway architectures are mostly based on industry standard servers (x86-based) and use open software stacks, which makes the development of a kit feasible. Development times for express benchmarks are much shorter compared to those for enterprise benchmarks.

4.2 Experiments

We ran our tests against HBase 1.2.0 [79, 80], which is a good representative system for this workload. Key-value stores for IoT data are being used in a variety of products and services, such as Google Cloud IOT [81] and Microsoft Azure IoT [82].

Our benchmarks are run on a cluster of eight Cisco UCSB-B200-M4 blade servers, each with the following configuration:

- 2 Intel(R) Xeon(R) CPU E5-2680 v4 clocked at 2.40GHz, each with 14 cores and 28 threads
- 256 GB RAM
- 2 Cisco UCS 6324 FI (10 GBps per server node)
- 2 Samsung 3.8 TB 2.5-inch Enterprise Value 6G SATA SSD

We used the following tuning parameters for HBase:

- hbase.client.write.buffer=8GB
- hbase.regionserver.handler.count=224
- Maximum number of Write-Ahead Log (WAL) files=128
- hbase.hstore.blockingStoreFiles=28
- Java Heap Size of HBase RegionServer=32GB
- Client Java Heap Size=8GB

This section is intended to show the basic performance characteristics of TPCx-IoT. However, it does not intend to showcase the best performance of the HBase nor analyze it in detail. The above tuning steps for HBase follow the best practices.

4.2.1 Scaling the Number of Power Substations

In order to determine how many power substations our gateway system supports, while still fulfilling the execution rule requirements outlined in Section 4.1.2, we ran experiments varying the number of power substations (number of TPCx-IoT

Table 4.2.1: Experiment parameters & requirement fulfillment

Power sub-stations	**Rows Ingested [Million]**	**Elapsed Time [s]**		**Ingestion Rate $\frac{kvps}{s}$**	
		Warm-up	**Mea-sured**	**System-Wide**	**Per-Sensor**
1	50	4,795	5,099	9,806	49.0
2	60	2,024	2,222	26,999	67.5
4	100	1,813	1,812	56,822	71.0
8	240	2,606	2,837	84,602	52.9
16	400	2,822	2,986	133,940	41.9
32	400	1,897	2,149	186,109	29.1
48	400	1,992	2,188	182,815	19.0

instances) from 1 to 48. We increased the number of power substations in steps of power of 2, except from 32-48, where we added 16.

Table 4.2.1 lists the benchmark input parameters, number of power-substations and number of *kvps* to be ingested in the first two columns. As we increased the number of power substations from 1 to 48, we increased the number of rows ingested from 50 Million to 400 Million to keep the elapsed times of the warmup and measured executions of the workload larger than 1800 s (columns three and four). Finding a suitable number of rows to ingest was not difficult. We binary-searched a suitable number for one power substation and extrapolated to multiple power substations. The system-wide throughput, which corresponds to the main metric ($IoTps$) of TPCx-IoT, increases from 9,806 $\frac{kvps}{s}$ with one power substation to 186,109 $\frac{kvps}{s}$ with 32 power substations. Adding 16 more power substations keeps the throughput constant at about 182,815 $\frac{kvps}{s}$. The per-sensor ingestion rate requirement of $20\frac{kvps}{s}$ can be fulfilled up to 32 power substations. At 48 substations it falls to $19\frac{kvps}{s}$.

The following graphs show additional metrics of the above conducted experiments. Unless noted otherwise, the number presented are measured, not derived. Figure 4.2.1 plots the system-wide throughput ($IoTps$). For each data point, we display the scaling number s based on the throughput with one power substations, that is, $S_i = \frac{IoTps_i}{IoTps_1}$ with $i \in 2, 4, 8, 16, 32, 48$ being the number of power substations. $IoTps$ scales super-linear until eight power substations. With two power substations S_2 is 2.8, with 4 power substations S_4 is 5.5 and with 8 power substations S_8 is 8.6. With 16 and more power substations $IoTps$ scales sub-linearly at $S_{16} = 13.7$, $S_{32} = 19.0$

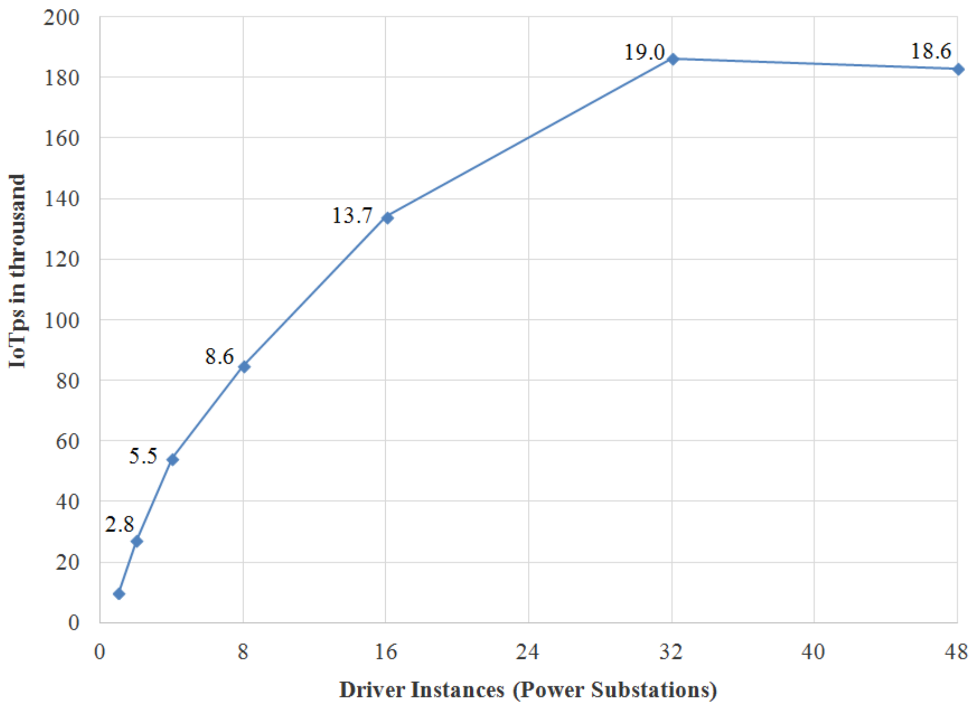


Figure 4.2.1: System-wide *kvp* inserted per second [IoTps]

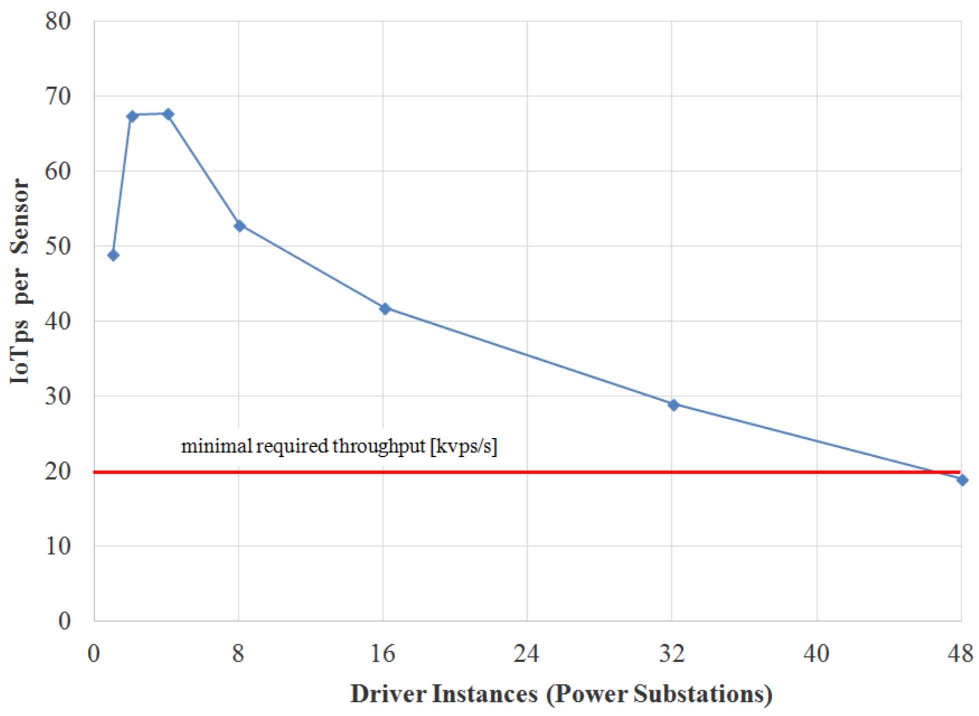


Figure 4.2.2: Per sensor *kvp* inserted per second

and $S_{48} = 18.6$.

Figure 4.2.2 plots the measured average per-sensor $IoTps$. The red line at 20 $IoTps$ indicates the minimum allowed for a valid benchmark run. $IoTps$ increases from 49.0 $IoTps$ with one power substation to 67.8 $IoTps$ with 4 power substations. Increasing the number of power substations to 48 causes $IoTps$ to drop below the allowable limit. The reason for the initial increase and subsequent drop in per-sensor throughput is the initial super-linear system-wide throughput scaling, which turns into a sub-linear scaling with 16 power substations. As the throughput per sensor is calculated by dividing the system-wide throughput by the number of sensors, and since the number of sensors increases linearly with the number of power substations, there is a direct correlation between the system-wide scaling and the per-sensor throughput: Sub-linear system-wide scaling causes a decrease in per-sensor throughput, linear system-wide scaling causes constant per-sensor throughput, and super-linear system-wide scaling causes an increase in per-sensor throughput.

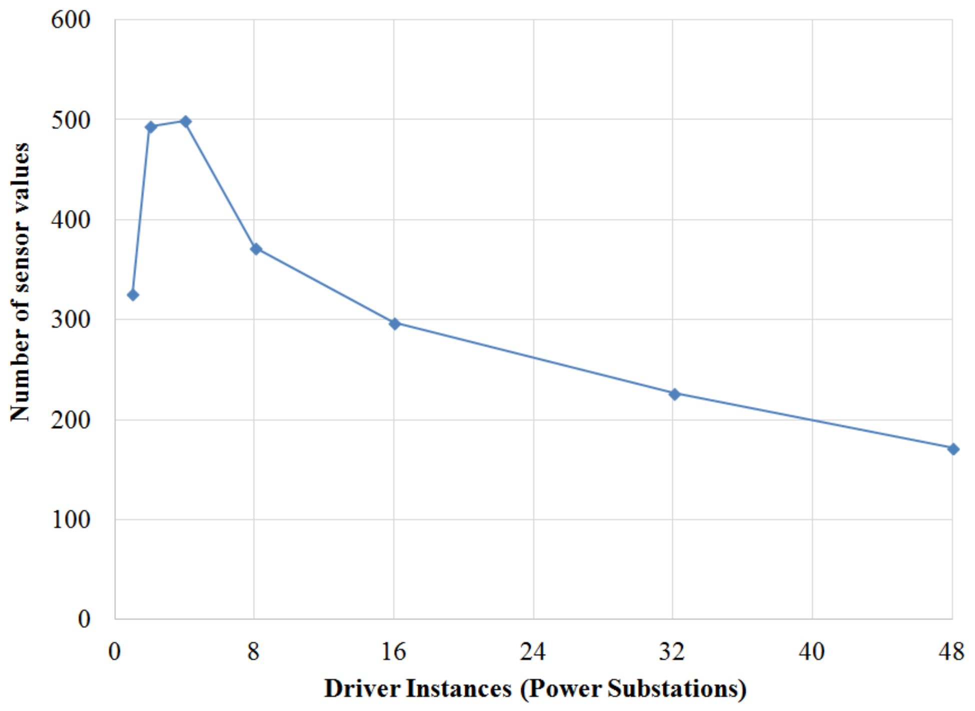


Figure 4.2.3: Average sensor readings aggregated per query

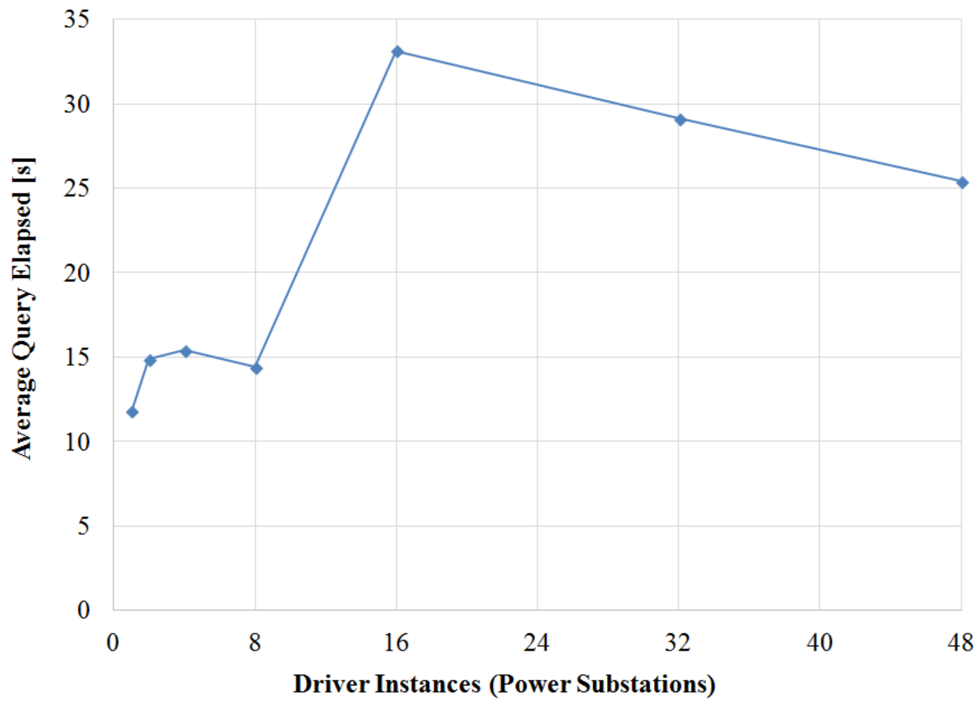
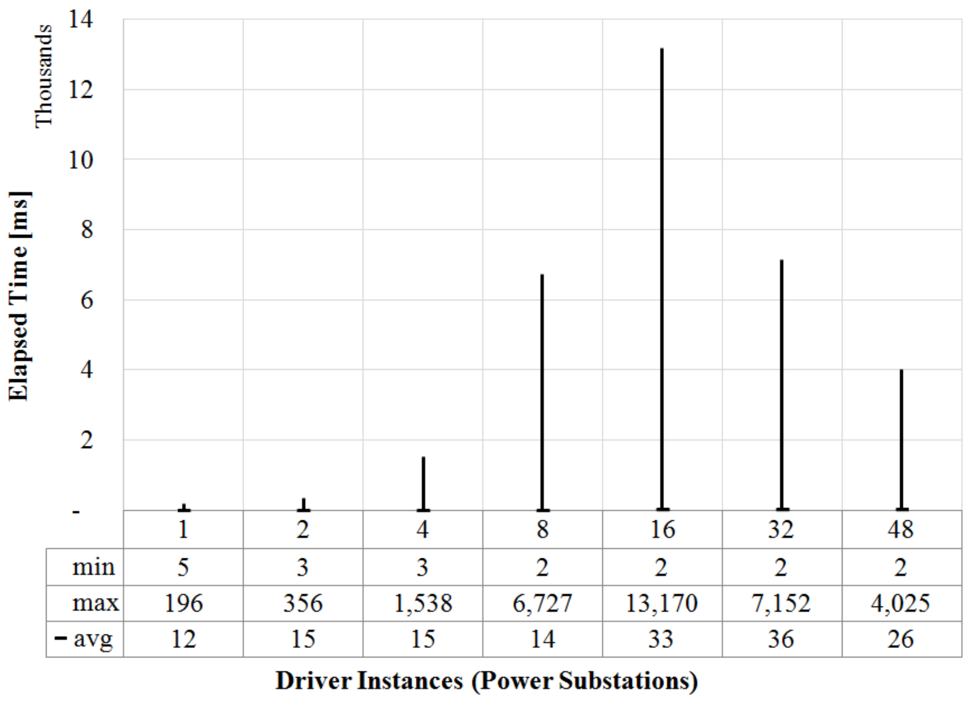


Figure 4.2.4: Average system-wide query elapsed time [s]

Figure 4.2.3 plots the measured average number of *kvps* aggregated by each query. This is essentially the average number of readings read per query for calculating the dash-board value for each sensor. The curve looks very similar to that of the previous Figure 4.2.2, because the number of data aggregated has the same correlation as the *Iotps* per sensor. We have included this graph to show that a reasonable number of values is considered while calculating the aggregate for each of the query template. If the number drops below 200, the benchmark run becomes invalid.

Figure 4.2.4 plots the average system-wide query elapsed time for each of our runs. Average query elapsed time is between 11.8 ms and 14.4 ms with up to 8 power substations. With 16 power substations, average query elapsed time increases to 33.1 ms and reduces slightly to 29.1 ms with 32 power substations and 25.4 ms with 48 respectively.

The bar chart in Figure 4.2.5 shows the minimum, maximum and average query elapsed times in milliseconds. For each number of power substations, the minimum

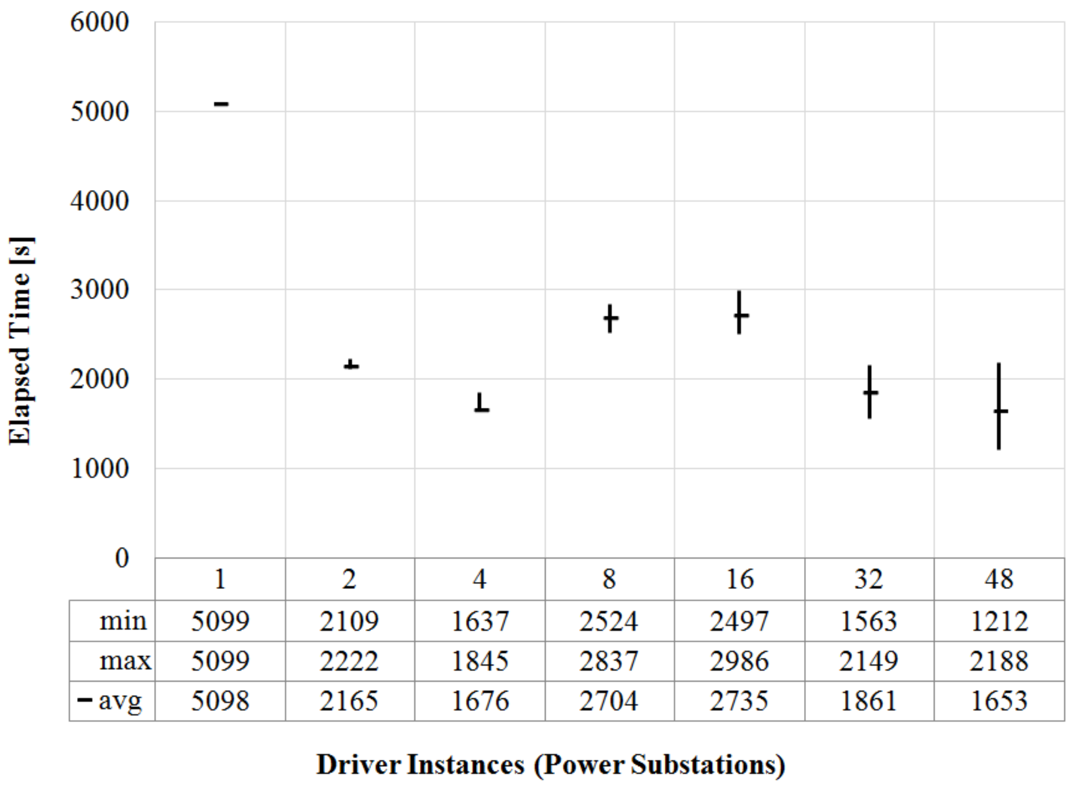


	1	2	4	8	16	32	48
min	5	3	3	2	2	2	2
max	196	356	1,538	6,727	13,170	7,152	4,025
- avg	12	15	15	14	33	36	26

Figure 4.2.5: Query elapsed time variation

and average query elapsed times are in double digit milliseconds (max is 36 milliseconds). However, the maximum query elapsed times are very high starting with 4 power substations (larger than 1000 ms). The numbers above each bar represents the coefficient of variation ($\frac{stdev}{mean}$). For each of the runs, the coefficient of variation is larger than 1, which indicates a very large variation. We also calculated the 95 percentiles for each run (not shown in figure). They were below 25 ms up to 16 power substations. They increased to 185 ms with 32 and 143 ms with 48 power substations. The benchmark currently does not require a query elapsed time percentile. However, due to the dash-board like use-case of TPCx-IoT, this is something to consider.

As TPCx-IoT requires the complete ingestion of a fixed number of sensor readings *kvps* from each power substation, as opposed to run for a fix amount of time, the benchmark tests the gateway system's ability to load-balance the data ingestion between all power substations. Figure 4.2.6 plots the minimum, maximum, and average ingestion times for data from all power substations (see Table 4.2.2 for a listing of the numbers). Although with one power substation all three values are



	1	2	4	8	16	32	48
min	5099	2109	1637	2524	2497	1563	1212
max	5099	2222	1845	2837	2986	2149	2188
– avg	5098	2165	1676	2704	2735	1861	1653

Figure 4.2.6: Power substation workload variation

identical, we include them for completeness. As we increase the number of power substations, the difference between the fastest and the slowest ingest time increases from 5% to 81%. Table 4.2.2 shows the times for the fastest ingest in the second column, the slowest ingest time in the third column and the average ingest time in the fourth column for each of the runs. The difference between the fastest and slowest are printed in Column 4 (absolute) and Column 5 (relative). For instance, the large difference of 81% in the 48 power substation case indicates that there is large potential for speeding this run up.

Table 4.2.2: Difference between fastest and slowest power substation ingest time

Power substation	**Ingest Time [s]**			**Difference**	
	Min	**Max**	**Avg**	**Absolute**	**Relative**
1	5,099	5,099	5,099	n.a.	n.a.
2	2,109	2,222	2165	113s	5%
4	1,637	1,845	1,676	208s	13%
8	2,524	2,837	2,704	313s	12%
16	2,497	2,986	2,735	489s	20%
32	1,563	2,149	1,861	586s	38%
48	1,212	2,188	1,653	976s	81%

4.2.2 Scaling the Number of Gateway Nodes (Scale-Out)

For the following set of experiments, we scaled the number of HBase servers from two to eight nodes. Due to the replication requirements in TPCx-IOT, the minimum number of nodes eligible for benchmark publication is two. All experiments conducted in this section fulfilled the runtime requirement of a minimal workload execution time of 1,800 s.

Figure 4.2.7 shows how TPCx-IoT throughput [*IoTps*] scales with the number of power substations on all three configurations. The corresponding data is also listed in Table 4.2.3. With the 2-node configuration (blue graph with rhombus markers), system-wide throughput increases from 21,909 *IoTps* with one power substation to 105,877 *IoTps* with eight power substations. Increasing the number of substations beyond eight increases the system-wide throughput to a peak of 115,486 *IoTps*. The 4-node configuration shows 15,706 *IoTps* with one power substation (red graph with square markers). This is about 28% less throughput than the two-node configuration. The system-wide throughput increases to 125,603 *IoTps* with 16 power substations. Increasing the number of power substations beyond 16 increases the system-wide throughput to a peak of about 134,248 *IoTps*. The 4-node configuration delivers a 16% higher peak system-wide throughput. As discussed in Section 4.2.1 the 8-node configuration scales to 32 power substations, after which the performance remains stagnant. At 9,806 *IoTps*, the 8-node configuration delivers the lowest performance with one power substation. However, with 182,815 *IoTps*, it can deliver the highest peak system-wide throughput.

Table 4.2.3: System-wide and per-sensor throughput for 2,4 and 8 nodes

Number of Power substations	**Throughput [IoTps]**					
	System-wide			**Per-sensor**		
	2-node	**4-node**	**8-node**	**2-node**	**4-node**	**8-node**
1	21,909	15,706	9,806	109.5	78.5	49.0
2	38,939	33,612	26,999	97.3	84.0	67.5
4	63,076	57,113	54,201	78.8	71.4	67.8
8	105,877	90,160	84,602	66.2	56.4	52.9
16	114,508	125,603	133,940	35.8	39.3	41.9
32	114,764	132,100	186,109	17.9	20.6	29.1
48	115,486	134,248	182,815	12.0	14.0	19.0

Figure 4.2.8 shows the per-sensor throughput for the three configurations. Similarly as the 8-node configuration, the 4-node configuration shows an initial increase in per-sensor throughput from one to four power substations. Per-sensor throughput begins to decrease after that. It stays above the limit of 20 *IoTps* until shortly after 32 power substations. The 2-node configuration does not show an increase in per-sensor throughput from one to two power substations, because, unlike the other configurations, the 2-node configurations shows sub-linear scaling starting from one to two power substations. Per-sensor throughput decreases steadily, starting from one power substation to 48 power substations. It crosses the limit of 20 *IoTps* exactly at 32 power substations.

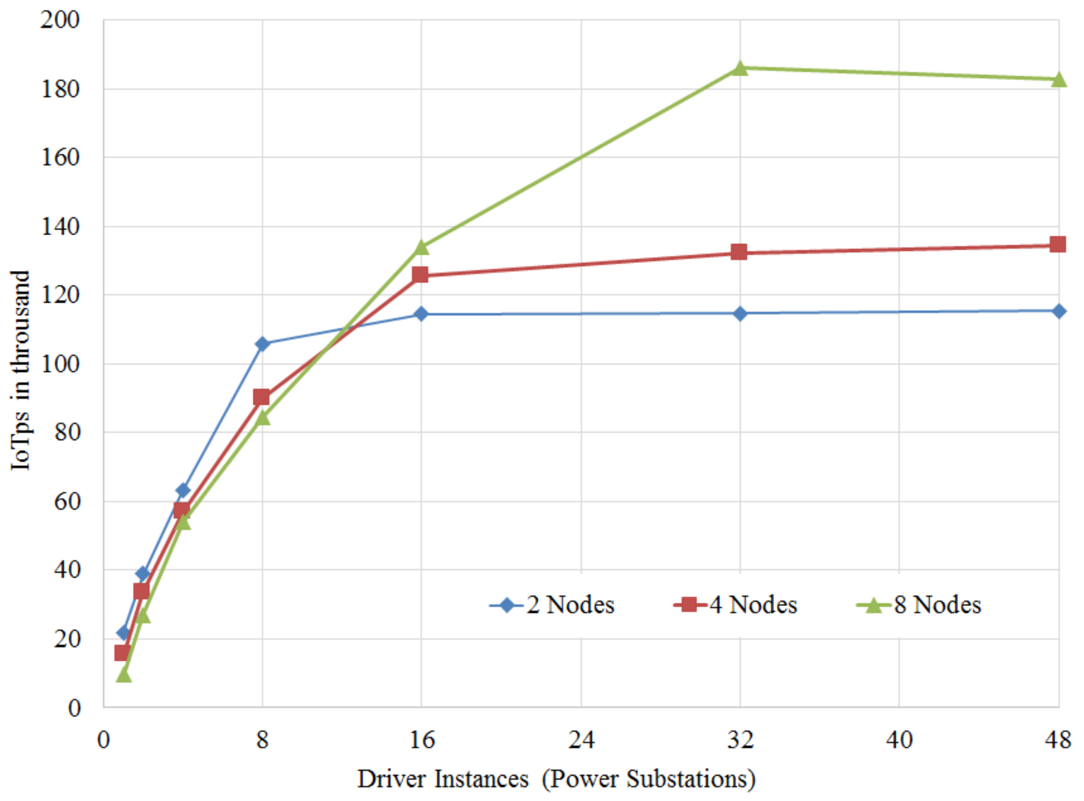


Figure 4.2.7: System-wide *kvp* inserted per second [IoTps]

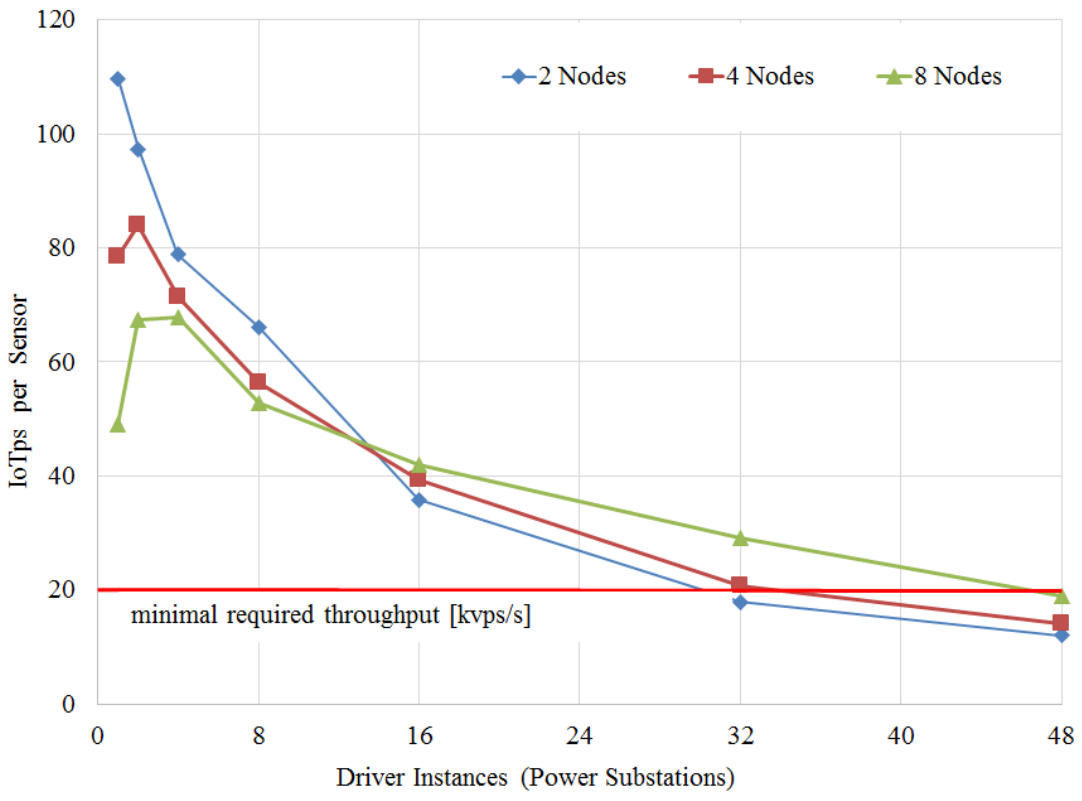


Figure 4.2.8: Per Sensor *kvp* inserted per second

Chapter 5

TPC-DI: First Industry Standard Benchmark for Measuring the Performance of Data Integration Systems

The term data integration (DI) covers a variety of scenarios, predominantly data acquisition for business intelligence, analytics and data warehousing, but also synchronization of data between operational applications, data migrations and conversions, master data management, enterprise data sharing and delivery of data services in a service-oriented architecture context, amongst others. Each of these scenarios requires the extraction of data from one or multiple source systems and data transformation and writing the data to one or more target systems.

While small DI deployments tend to be implemented using collections of customized programs or database procedures, medium to large sized DI deployments are usually implemented using general purpose DI tools. These deployments often must integrate data from many disparate data sources with various formats requiring complex formatting and data transformations prior to loading into one or more target systems. General purpose DI tools can significantly increase developer productivity by providing commonly used functionality for system connectivity and for standard data transformations. They further improve the availability and maintenance of the DI processes by visualizing connections, transformations and progress of running tasks. A non-exhaustive list of commercially available tools includes, for example, Ab Initio[1], IBM InfoSphere Information Server for Data Integration[2] Microsoft SSIS[3], and Oracle Warehouse Builder[4].

Ever since vendors started implementing and marketing general purpose DI tools, they started making competitive and performance claims. With no standard DI benchmark available, the situation is similar to that of 1980s, when many system vendors due to the the lack of standard database benchmarks practiced what is now referred to as “benchmarketing”. A large number of *world record* claims have been made for DI systems (for example, [83, 47, 48]). These are of no value to customers who would like to evaluate DI performance across vendors. Having realized this void the Transaction Processing Performance Council (TPC) released the first version of its data integration benchmark, TPC-DI, in January 2014[5]. TPC-DI is modeled using the data integration processes of a retail brokerage firm, focusing on populating a decision support system with transformed data from a variety of desparate systems, including a trading system, internal Human Resource (HR) and Customer Relationship Management (CRM) systems. The mixture and variety of operations being measured by TPC-DI are not designed to exercise all possible operations used in DI systems. And they are certainly not limited to those of a brokerage firm. They rather capture the variety and complexity of typical tasks executed in a realistic data integration application that are characterized by:

- The manipulation and loading of large volumes of data
- A mixture of transformation types including error checking, surrogate key lookups, data type conversions, aggregation operations, data updates, etc.

[1]`http://www.abinitio.com/`

[2]`http://www-03.ibm.com/software/products/en/infoinfoservfordatainte`

[3]`http://technet.microsoft.com/en-us/library/ms141026.aspx`

[4]`http://www.oracle.com/technetwork/developer-tools/warehouse/overview/introduction/index.html`

[5]`http://www.tpc.org/tpcdi/default.asp`

- Historical loading and incremental updates of a decision support system using the transformed data
- Consistency requirements ensuring that the integration process results in reliable and accurate data
- Multiple data sources having different formats, including, multi-row formats and XML
- Multiple data tables with varied data types, attributes and inter-table relationships

Because there is no standard language to express a data integration workflow and transformations, TPC-DI is implemented as an express class benchmark.

The remainder of this chapter is structured as follows. Section 5.1 introduces the source and target data model. Section 5.2 presents the characteristics of the data sets used to populate the source model and explains the technical details of how the data sets are generated and scaled. In Section 5.3, the transformations of the DI workload are explained. They form the core workload of the benchmark. Section 5.4 presents the execution rules that govern how the transformations have to be executed, timed, and weighted to compute the ranking of DI systems. Section 5.5 presents the metric and explains the decisions that led to its definition. A performance study is presented in Section 5.6.

5.1 The Data Model

The data model of TPC-DI is designed to exercise much of the functionality typically used in today's DI systems. It consists of two main parts, the *source data model* and the *target data model*. The source data model represents the input data set to the data integration process. It resembles data from online trading operations combined with other internal data sources, that is, a human resource and a customer management system, externally acquired data, financial newswire data, and customer prospect data. The target data model resembles a dimensional decision support system following common practice in the industry [29]. It consists of multiple fact tables sharing various types of dimensions. This snowflake schema variant enables easy and efficient responses to typical business questions asked in the context of a retail brokerage firm. There are other ways to define a decision support system, but this format provides a well understood structure in the benchmark while also allowing for an appropriate variety of data transformations to be exercised in the core workload.

Figure 5.1.1 outlines the conceptual model of the TPC-DI benchmark. The top portion displays the six data sources. While in real world scenarios it is necessary to extract data from these sources including different database vendors and file structures, the actual extraction from physical systems of these types is out of scope of the benchmark. While it would be desirable to include the extraction from these often heterogeneous source systems, it is an intractable problem from a benchmark logistics point of view given the technology agnostic specification of the benchmark. And it is often forbidden in the end-user license agreement of commercially available products. Hence, TPC-DI models an environment where all source system data has been extracted into flat files in a staging area before the timed portion of the DI process begins. TPC-DI does not attempt to represent the wide range of data sources available in the marketplace, but models abstracted data sources and measures all systems involved in moving and transforming data from the staging area to the target system. The use of a staging area in TPC-DI does not limit its relevance as it is common in real world DI applications to use staging areas for allowing extracts to be performed on a different schedule from the rest of the DI process, for allowing backups of extracts that can be returned to in case of failures, and for potentially providing an audit trail. The following two subsections describe the source and target data models more in detail.

The lower part of Figure 5.1.1 shows the benchmarked system, commonly referred to as the system under test (SUT). It consists of three conceptually different areas, which may reside on any number of physical or logical systems. The staging area holds the source data that is read by the data transformations. No manipulations are allowed on the files after they are generated by the data generator and placed into the staging area. This guarantees that all methods used to speed up the execution of the data transformations, for example, sorting the data or splitting data into multiple files, are performed in the following timed portion of the benchmark. The data transformations read the source data and perform all necessary modifications so the target system can be populated. The data transformations are described in detail in Section 5.3. The target system can be a business intelligence, a data warehouse, a business analytics system, or a master data management system. We refer to it as the *decision support system*.

5.1.1 Source Data Model

Typical DI applications support two integration processes with different characteristics and performance requirements. One process performs an initial load of the target system, the *historical load*. A second process performs periodic trickle updates into it, that is, *incremental updates*. The concepts of historical load and incremental updates are described comprehensively in Section 5.5. For the most part, the general structures of the data models for these two concepts are identical. However, there are differences in the use of input files in each of the two types of load. In the remainder of this section, we will introduce the various input files, their purpose in the context of the DI process, what part of transformations they enable, how they are populated, and how they scale.

As mentioned above, TPC-DI's source data model is based on internal data of the operational system of the fictitious retail brokerage firm, externally acquired marketing data and reference data. The operational system is comprised of an online transaction processing database (OLTP DB) for the online trading department system, a human resource system (HR) and a customer relationship management system (CRM). The externally acquired data is comprised of financial data (FINWIRE) of publicly traded companies, delivered by a newswire system, and customer prospect data, acquired through a marketing firm (PROSPECT). The reference data contains static information, only required to be loaded during the historical load, such as date/time, industry segments, tax rates, and trade types.

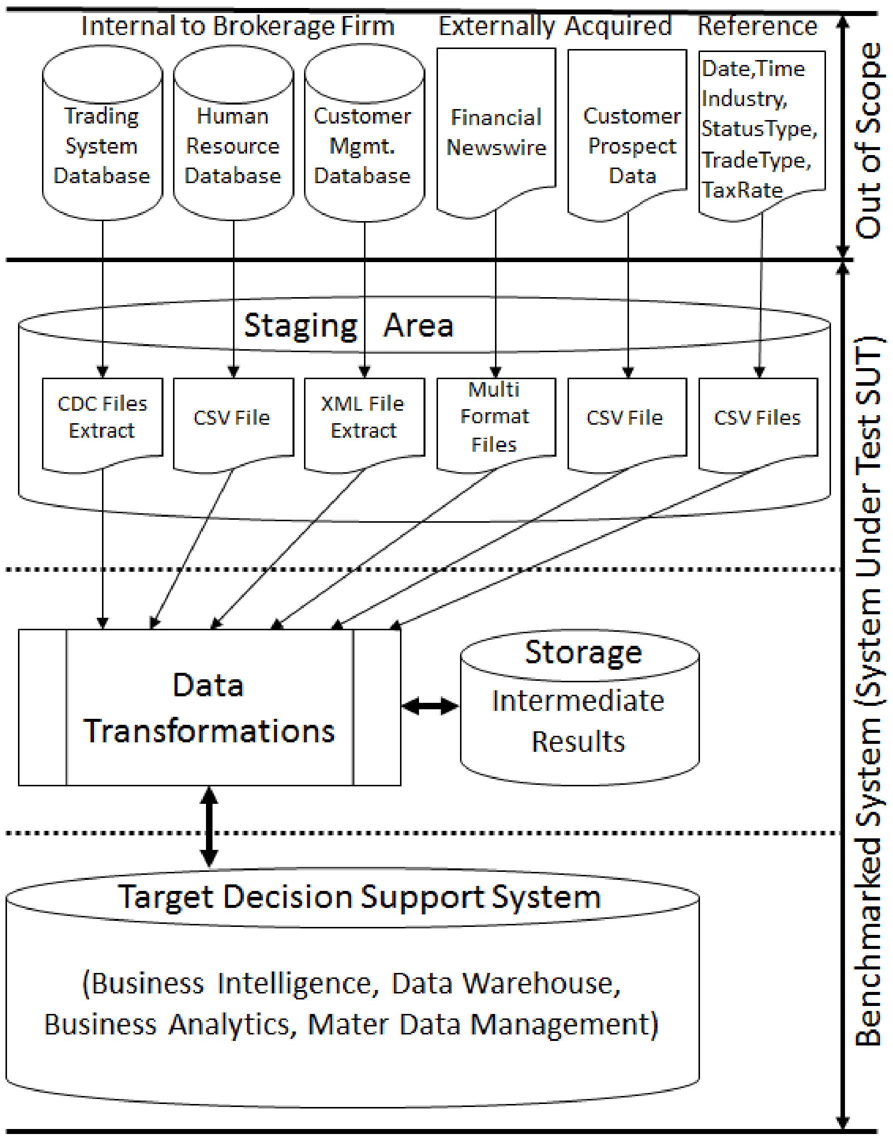


Figure 5.1.1: Benchmarked system and workflow

The OLTP DB represents a relational database with transactional information about securities market trading. It contains the following tables: (i) customers, (ii) accounts, (iii) brokers, (iv) account balances, (v) securities, (vi) trade details, and (vii) market information. Files used in the historical load are full extracts containing all rows in the corresponding table of the source system. Files used in the incremental update are change data capture (CDC) extracts, and as such they contain additional flags, that is, CDC_FLAG and CDC_DSN columns at the beginning of each row. The CDC_FLAG is a single character I, U or D that tells whether the row has been inserted (I), updated (U) or deleted (D) since the previous state. For updates there is no indication as to which values have been changed. Rows that have not changed since the last extract will not appear in the CDC extract file. A row may change multiple times in the course of a day[6]. In this case, the DI process needs to merge all change records to determine the values of the record to be inserted. The CDC_DSN is a sequence number, a value whose exact definition is meaningful only to the source database, but is monotonically increasing in value throughout the rows in a file. The rows in a file are ordered by the CDC_DSN value, which also reflects the time order in which the changes were applied to the database.

The HR system contains employee data of the fictitious retail brokerage firm including employee name, job description, branch location, contact information, and management chain. The HR database is represented by a single extract file, HR.csv. There is no CDC on this data source; it is modeled as a full table extract for the historical load.

The CRM system, an OLTP source, contains customer contact information and information about their accounts. Data from this system is presented in form of an XML file. Its structure is hierarchical to represent data relationships between customers and their accounts. Each record in this file represents an action performed in the CRM system, that is, *New* (new customer), *AddAcct* (add a new account), *UpdAcct* (update an existing account), *UpdCust* (update an existing customer), *CloseAcct* (close an existing account),*Inact* (inactivate an existing customer). This data is only used in the historical load.

[6]day is the refresh interval for incremental updates

Data for the two external sources are also represented by file extracts. *Prospect* represents data that is obtained from an external data provider. Each file contains names, contact information and demographic data of potential customers. Since the data is coming from an independent source, it cannot be guaranteed that it is duplicate free, that is, some person in the prospect file might already be a customer of the brokerage firm. The DI tool needs to account for duplicates. This file is modeled as a full daily extract from the data source. This also means that there is no indication as to what has changed from the previous extract.

Finwire data represents financial records from companies that have been recorded over three month periods. Data for each three month period is grouped together in one file, for example FINWIRE2003Q1 for data of the first quarter of 2003. Each of these files can contain records of the following type CMP = company, SEC = security, FIN = financial. Each record type has its own distinct schema. The type of record in this variable length data extract is indicated in the first three bytes.

The reference data provided in, Date.txt, Time.txt, Industry.txt, StatusType.txt, TaxRate.txt and TradeType.txt is loaded only during the historical load. While one expects these tables to change in the lifetime of a real-world system, they are kept static in TPC-DI.

Data from the above described sources is generated by a TPC provided data generator, DiGen, which is implemented using PDGF, the Parallel Data Generation Framework, developed at the University of Passau [30]. More on the data generator in Section 5.2.3.

Table 5.1.1 summarizes all source input files used during the historical and incremental loads. The first column denotes the file name, the second the file format. The third and fourth columns denote whether a file is used as input for the historical or/and incremental load phases.

Source Table	**Format**	**H**	**I**
Account.txt	CDC		✓
CashTransaction.txt	DEL/CDC	✓	✓
Customer.txt	CDC		✓
CustomerMgmt.xml	XML	✓	
DailyMarket.txt	DEL	✓	✓
Date.txt	DEL	✓	
Time.txt	DEL	✓	
FINWIRE	Multi-record	✓	
HoldingHistory.txt	DEL	✓	✓
HR.csv	CSV	✓	
Industry.txt	DEL	✓	
Prospect.csv	CSV	✓	✓
StatusType.txt	DEL	✓	
TaxRate.xt	DEL	✓	
TradeHistory.txt	DEL	✓	
Trade.txt	DEL/CDC	✓	✓
TradeType.txt	DEL	✓	
WatchItem.txt	DEL/CDC	✓	✓

Table 5.1.1: Source files with type (DEL=full data dump, CDC=change data capture, XML=Extensible Markup Language, CSV=Comma Separated Value) and load usage

5.1.2 Target Data Model

The target data model is organized as a snowstorm schema, an extension to the well-known star schema. It is similar to that deployed in TPC's latest decision support benchmark, TPC-DS [84]. In general, a star schema includes a large fact table and several small dimension (lookup) tables. The fact table stores frequently added transaction data such as security trades and cash transactions. Each dimension table stores less frequently changed or added data supplying additional information for fact table transactions, such as customers who initiated a trade. An extension to the pure star schema, the snowflake schema, separates static data in the outlying dimension tables from the more dynamic data in the inner dimension tables and the fact tables. That is, in addition to their relation to the fact table, dimensions can have relations to other dimensions. Combining multiple snowflake schemas into one schema results in a snowstorm schema. Usually, fact tables of a snowstorm schema share multiple dimensions. In many cases joins between two or more fact tables are possible in a snowstorm schema making it very interesting for writing challenging queries and transformations.

The design goal for the TPC-DI target schema is to realistically model what real world customers currently use as part of their data integration processes. There are other ways to define a decision support system, but the snowstorm model provides a well understood structure while also allowing for an appropriate variety of data transformations to be exercised as part of the main task in TPC-DI. Figure 5.1.2 shows a simplified ER diagram of the target data schema.

TPC-DI defines seven dimension tables:(i) Date, (ii) Time, (iii) Customer, (iv) Account, (v) Broker, (vi) Security, and (vii) Company. These dimensions provide details for six fact tables: Holding, (i) Trade, (ii) Cash Balances, (iii) Market History, (iv) Watches, and (v) Prospects. The schema also includes five reference tables that have no relation to any of the fact or dimension tables. Their purpose is to provide additional information during the transformation. These are: (i) Trade, (ii) Type, (iii) Status Type, (iv) Tax Rate, (v) Industry, and (vi) Financial.

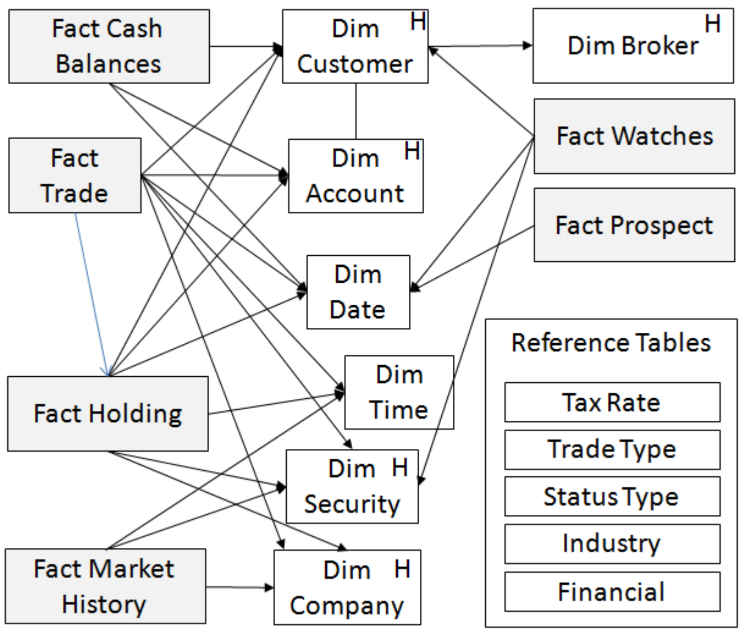


Figure 5.1.2: Target data schema

5.2 Data Set

The data set for a particular benchmark is driven by the need to challenge the performance of all components it measures, hardware and software. In the case of TPC-DI, these are reading and interpreting of source data from a staging area, and data transformations and data loading into the target decision support system for both the historical and incremental load phases. In this context, a well designed data set stresses the statistic gathering algorithms, the data interpretation and transformation engines, data placement algorithms, such as clustering, vertical or horizontal partitioning as well as insert strategies for bulk and trickle loads. A good data set design includes proper data set scaling, both domain and tuple scaling. Like in other TPC benchmarks, a hybrid approach of domain and data scaling is used for TPC-DI. The data domains for tables are very important. While pure synthetic data generators have great advantages, TPC-DI follows a hybrid approach of both synthetic and real world based data domains. Synthetic data sets are well understood, easy to define and implement. However, following the TPC's paradigm to create benchmarks that businesses can relate to, a hybrid approach to data set design has many advantages over both pure synthetic and pure real world data.

5.2.1 Real World Relevance of the Data Set

The data set used in TPC-DI resembles very closely that of a real brokerage firm. This complex data set requires a sophisticated data generator that is able to create patterns apparent in real life data sets, that is, address changes occurring in a certain time order, trade transactions that affect multiple accounts, such as account balances and security holdings, or trades that go through a series of states from placement to fulfillment. These data characteristics can be formalized as intra row, intra table and inter table dependencies [85]. Intra row dependencies occur when some fields of the same row exhibit some sort of dependencies. For instance, in the US, value added tax (VAT) varies by state and within some states by county. Hence, the VAT depends on the location of the purchase. Intra table dependencies occur when values of different rows within the same table have dependencies as it often occurs in history-keeping dimensions. Inter table dependencies occur if rows in different tables need to be related to each other, like for referential integrity when multiple tables are updated as part of an event, for example, a security trade.

The following paragraphs illustrate the complexity of TPC-DI's data set and its real world relevance using security trades as an example. Securities are equities or debentures of publicly traded companies that fluctuate in value over time. Trading securities can either be an equity (cash account) or debenture (margin account) and is done usually via a brokerage firm, either through a registered representative or without a broker through an online brokerage trading firm. Cash accounts require all transactions to be paid for in full by the settlement date three days after the trade execution. Margin accounts allow the investor to borrow money for the purchase of securities in hopes that they will not go down in price and a margin call for the difference is demanded by the brokerage firm. TPC-DI models security trades fulfilled by a cash account and done by a registered representative brokerage firm.

When trades occur four tables are affected. Rows in the *trade*, *trade history*, *holding history*, and *cash transaction* tables are tightly interconnected using all three of the above mentioned type of data dependencies. Additionally, the content of other input tables, for example, security.txt and customer.txt need to be consulted to assure that only valid customers trade existing securities, which is not trivial as

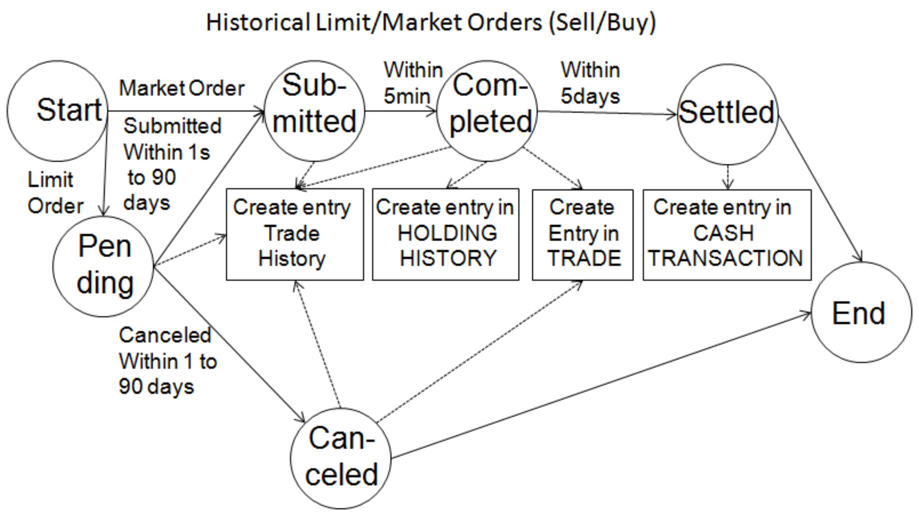

Historical Limit/Market Orders (Sell/Buy)
Start
Market Order
Submitted Within 1s to 90 days
Limit Order
Sub-mitted
Within 5min
Com-pleted
Within 5days
Settled
Pen ding
Create entry Trade History
Create entry in HOLDING HISTORY
Create Entry in TRADE
Create entry in CASH TRANSACTION
End
Canceled Within 1 to 90 days
Can-celed

Figure 5.2.1: State diagram for the order data creation of the historical load

new customers and security symbols are added over time. The trade table contains information about each customer trade. The holding contains information about customer securities holding positions that were inserted, updated, or deleted and which trades caused each change in holding. The cash transaction table holds data about cash transaction of customer accounts. These cash transaction usually follow a trade fulfillment. Figure 5.2.1 shows the state diagram of trades for the historical load. An order enters the system either as a *market order* or a *limit order*. Market orders are executed at the current market price. Limit orders are executed at the price specified or canceled. Market order transitions create an entry in the Trade History table immediately after they enter *submitted* state. Within five minutes they transition to *completed* where they create another entry in the Trade History table, an entry in the Holding History table and in the Trade table. Within five days they transition to *settled* where they create an entry in the Cash Transaction table. Limit orders on the other hand transition immediately after they are placed to the *pending* state where they create an entry in the Trade History table. Then they transition either to the *submitted* state or they transition to the *canceled* state. When they transition to *submitted* they follow the path of the *market order* or to *canceled* where they create entries in the Trade History and Trade tables.

5.2.2 Data Set Scaling

Being able to scale a data set is pertinent for any benchmark because of two main reasons. Firstly, for a benchmark to be relevant to real world problems, it needs to reflect data sizes used in real world systems. Ultimately, the *customers* of TPC-DI benchmark results are end-users trying to evaluate the performance and price performance of DI solutions. Customer data sets tend to vary greatly from one business to another and, therefore, those who publish benchmark results must be able to size their benchmark publication to the customers they are catering to. Secondly, systems and data sets tend to grow rapidly over time. A benchmark with a static data set size will become obsolete within a few years due to the compute power used by real world applications. Hence, a benchmark needs to be able to adapt to different data sizes.

Data set scaling has two orthogonal aspects, determining the cardinality of each individual relation of a schema based on a common scale factor SF and expanding a base data set to reach the cardinalities desired. Using the same scale factor SF to determine all table cardinalities helps in creating a coherent data set. Additionally, using the cardinality of a particular entity modeled in the data set as the scale factor SF helps understanding the data size resulting from a particular scale factor, for example, number of customers or number of ticker symbols. There are two approaches in defining SF:(i) continuous scaling, that is, $SF \in \mathbb{N}$, or (ii) fixed scaling, that is, a limited number of predefined scale factors $SF \in \{C_1, C_2, ..., C_n\}$. Continuous scaling requires that performance of results obtained from different scale factors are comparable. "Comparable" in this context means that the workload scales linearly, that is, data sizes and amount of work required by transactions. It is understood that not all algorithms scale linearly with data sizes and work required by transactions. However, for the purpose of comparing results with continuous scaling it is sufficient that the following is met: Assuming throughput metric $P_{S,BM}(SF)$ when run system S using benchmark BM, then $P_{S,BM}(SF) = \epsilon * P_{S,BM}(SF')$ for small increments from SF to SF'. Fixed scaling avoid this issue by only requiring comparability of results obtained with the same scale factor.

TPC-DI uses continuous scaling based on the number of customers of the fictitious

Source Table	Size in Bytes	Number of rows
Date.txt	3372643	25933
Time.txt	4060800	86400
Industry	2578	102
StatusType	83	6
TaxRate.	16719	320
TradeType	94	5

Table 5.2.1: Reference source files size and rowcount information

brokerage firm. The number of unique customers UC_H that are present in the historical data set can be computed as $UC_H(SF) = SF * 5000$. Each incremental load makes changes to or adds customers in the decision support system at a rate of $5 * SF$ customers per update.

Data set expansion can take on two different characteristics. In one case, the number of tuples in the base data set is expanded, but the underlying value sets (the domains) remain static. The business analogy here is a system where the number of customers remains static, but the volume of transactions per year increases. In the other case, the number of tuples remains fixed, but the domains used to generate them are expanded. For example, there could be a new ticker symbol introduced on wall street, or it could cover a longer historical period. Clearly there are valid reasons for both types of scaling within a dataset, just as there are valid reasons to stress a hardware or software systems to highlight particular features or concerns, and often a test will employ both approaches to expanding the dataset. As has been proven beneficial for other TPC benchmarks, such as TPC-DS, in the case of TPC-DI, the choice was made to use a hybrid approach. Most table columns employ data set expansion instead of domain expansion, especially fact table columns. Some columns in small tables employ domain expansion. The domains have to be scaled down to adjust for the lower table cardinality.

Source data for fact tables and most dimension tables scale linearly with SF. Therefore, the size $size_F$ of input file F at scale factor SF can be expressed as $size_F(SF) = SF * S_F$, with S_F being a factor specific for table F. Similarly, we can express the number of rows $rows_F$ of input file F at scale factor SF as $rows_F(SF) = SF * R_F$. With R_F again being a factor specific for Table F. Other

Source Table	S_H	S_I	R_H	R_I
CashTransaction.txt	10.58	0.0065	120.30	0.0663
CustomerMgmt.xml	2.87	N.A.	107.66	N.A.
DailyMarket.txt	30.02	0.0499	541.55	0.7619
FINWIRE	9.70	N.A.	49.32	N.A.
HoldingHistory.txt	2.66	0.0023	120.47	0.0663
TradeHistory.txt	10.31	N.A.	326.56	N.A.
Trade.txt	12.56	0.0175	130.00	0.1801
WatchItem.txt	13.37	0.0383	300.00	0.6896
Account.txt	N.A.	0.0007	N.A.	0.0100
HR	0.3914	N.A.	5	N.A.
Customer	N.A.	0.0099	N.A.	0.0050
Prospect	N.A.	0.9958	N.A.	4.994

Table 5.2.2: Source files scaling information

tables, such as date and time, do not scale with SF, they remain static.

Table 5.2.2 summarizes the scaling of all source data files that scale with the scale factor both for the historical load (H) and incremental loads (I). Columns labeled S_H and S_I list the table specific factors to calculate the size [GB] and columns labeled R_H and R_I list the table specific factors to calculate the number of rows for both the historical load (H) and incremental loads (I). Table 5.2.1 summarizes the sizes of all static tables.

5.2.3 Data Generation with PDGF

Since its first incarnation, the parallel data generation framework PDGF, which was developed at the University of Passau [30], has been improved and extended with many features. Its portable and high performance data generation methods are very configurable and extensible, allowing the generation of data for any kind of relational schema, while hiding the complexity of parallel data generation on today's massive scalable systems and drastically reducing the development time for a data generator. All these features convinced the TPC to choose PDGF for the development of DiGen.

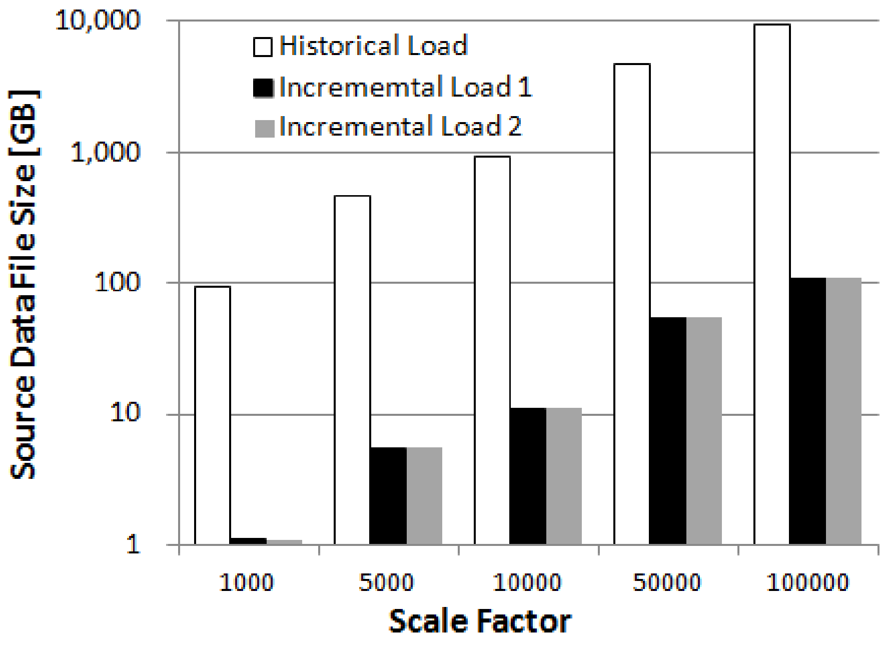


Figure 5.2.2: Aggregated source data sizes for historical and incremental loads in GB

Since 2013 PDGF is being commercialized by bankmark[7].

PDGF is configurable using two XML configurations files. And its rich plug-in system enables Java knowledgeable programmers to extend it very easily. Performance tests have shown that it is equally fast in generating TPC-H data as dbgen, TPC's C-based custom built reference implementation. It uses a special seeding strategy to exploit the inherent parallelism in pseudo random number generators. By incrementally assigning seeds to tables, columns, and rows the seeding strategy keeps track of the random number sequences for each value in the data set. This makes it possible to re-calculate values for references and correlations rather than storing them. This makes PDGF highly scalable on multi-core, multi-socket, and multi-node systems, that is, for scale-up and scale-out.

PDGF hides all reference, update, and general random number generation in an abstraction layer called update black box. Generic generators for numbers, strings, text, and references use the black box to get the correct random number sequences. The data generation itself is performed by worker threads that generate blocks

[7]`http://www.bankmark.de`

of data and optionally sort it using a cache. The generated data can be further formatted using a post-processing system that enables elaborate transformations of the generated data. Users specify the data model in form of an XML configuration file. The data model consists of tables, columns, and generators, which contain the semantic of the data model. Furthermore, users can specify transformations in a second XML file. These transformations can be simple formatting instructions, but also complex merging or splitting of tables.

Due to the complex dependencies in the TPC-DI specification additional forms of repeatable data generation had to be developed in PDGF. One of the biggest challenges was the generation of consistent updates to the historical load. An example is the table Customer. Customers can be inserted, updated, and deleted. While creating new customers in updates is relatively easy and is essentially the same process as writing the historical table, updates are written as full records, repeating historic or previously updated values. Also, updates and deletes cannot be generated for previously deleted records. To support this tracking of change, an abstract notion of time was introduced to PDGF [86]. In each abstract time step, a row or record can either be inserted, updated, or, deleted. A row's life cycle thus starts by its insertion, potentially followed by updates, and ends with its deletion. To keep track of the changes a set of permutations is used as described in [86].

One of the most complex parts of the TPC-DI data set is the model of security trades. The trades have a live cycle that is shown in Figure 5.2.1. The different states of trades are stored across multiple tables and these tables store the history of trades, meaning that all states have to exist and be consistent already in the historical tables. To achieve this level of detail at the required performance, a specialized update black box was implemented, which completely implements the trade life cycle. Essentially, all trade relevant information is modeled in a single table, which is split up during the generation. Technically, the data will not be split up during generation, but only the required values will be generated. To ensure the time consistency, all trade related tables are built of many updates, in which each record can be transferred into a new state. Depending on the time granularity of the tables the time unit is fraction of days to quaters of years.

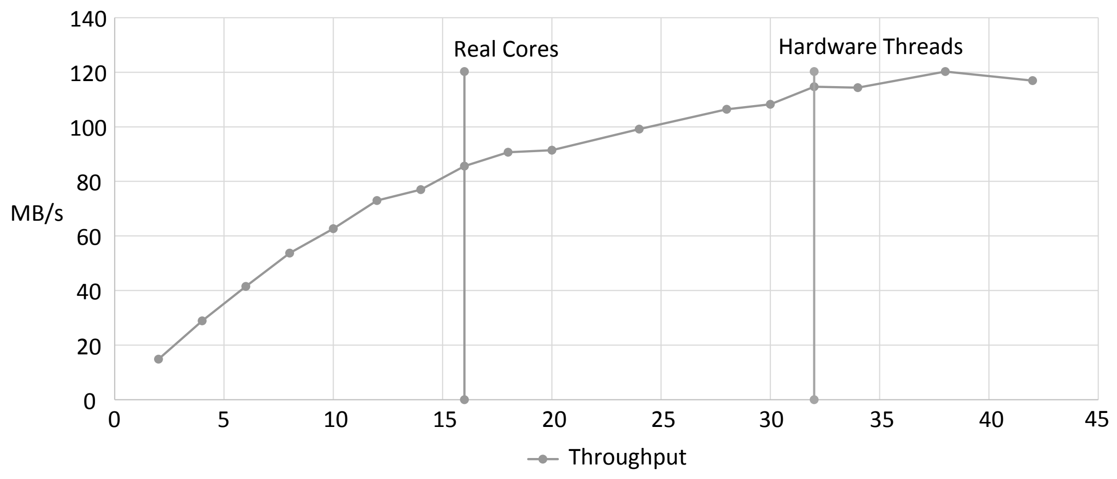


Figure 5.2.3: DiGen scale-out performance

PDGF supports all file formats required by TPC-DI, such as CSV, text, multi format, and XML. PDGF supports this using an output system that transforms data from row oriented data to any other representation. PDGF comes with several output plug-ins such as character separated value data (for example, CSV), XML formats, and a generic output that can be scripted using Java code. Internally, the workers generate blocks of data for each table or set of tables that is currently scheduled. The output system receives the internal representation and uses the output plug-in to transform the data. Besides the formatting, it is possible to merge or split tables in the output, although this functionality can also be achieved by changing the model, it is desirable to have a clean and understandable model and keep pure formatting separated. The output system enables separate formatting per set of tables, it is also possible to generate tables in multiple formats. Using a property system, the format can also be determined at run time.

Figure 5.2.3 shows the scale-out performance of DiGen. We generated data for scale factor 100 on a system with 2 E5-2450 Intel CPUs, that is, 16 cores and 32 hardware threads. The data was generated repeatedly by increasing the number of workers from 1 to 42. The generation scales almost linearly with the number of cores. Data generation continues to increase beyond the number of threads (32), but slows down after 38 threads.

5.3 Transformations

TPC-DI's transformations define the work that must be completed to prepare and load data into the data warehouse. In essence, they provide a mapping of data in the source tables to data in the target tables. TPC-DI defines two transformations for each of the fact and dimension tables of the target decision support system as described in Figure 5.1.2, one for the historical and one for the incremental loads. The transformations are not explicitly named, but since there are two for each target table, they can be referred to by using a combination of the name of the target table that they populate and name of their load phase. For instance, the transformation that populates the DimAccount table during the historical load is named $T_{H,DimAccount}$. Each transformation stresses particular characteristics of a DI system. While not all transformations cover disjunct characteristics, taken together, all transformation cover most work performed during typical DI transformations. Their characteristics are summarized in Table 5.3.1. The first column labels the characteristic so that we can refer to it later, the second column briefly describes it.

There are a total of 18 transformations defined, each of which is defined in English text. Unlike well established languages to describe query result sets, such as Structured Query Language (SQL) or XQuery, to date there is no common language to describe DI transformations. DI transformations are defined in terms of the data warehouse table(s) they populate. For each field of the data warehouse table(s), the source data field(s) and any transformations required to be performed on the source data are specified in English text. While it allows for a wide degree of freedom in implementing and optimizing the workload, it also imposes challenges to the benchmark specification to assure a "level playing field" for everybody. To guarantee that all benchmark sponsors interpret the English text in the same way, that is, get the same result and do not over-optimize or cut corners, TPC-DI defines a qualification test. It provides an input data set, that is, SF=5 DIGen data, and a corresponding dump of the decision support system after all transformations have been executed.The TPC-DI specification cannot provide qualification output for all scale factors because of its continuous scaling model. The qualification tests must be performed on the SUT using the same hardware and software components as the performance test and configured identically to those of the performance test. The content of the

Label	Description
C_1	Transfer XML to relational data
C_2	Detect changes in dimension data, and applying appropriate tracking mechanisms for history keeping dimensions
C_3	Update DIMessage file
C_4	Convert CSV to relational data
C_5	Filter input data according to pre-defined conditions
C_6	Identify new, deleted and updated records in input data
C_7	Merge multiple input files of the same structure
C_8	Convert missing values to NULL
C_9	Join data of one input file to data from another input file with different structure
C_{10}	Standardize entries of the input files
C_{11}	Join data from input file to dimension table
C_{12}	Join data from multiple input files with separate structures
C_{13}	Consolidate multiple change records per day and identify most current
C_{14}	Perform extensive arithmetic calculations
C_{15}	Read data from files with variable type records
C_{16}	Check data for errors or for adherence to business rules
C_{17}	Detect changes in fact data, and journaling updates to reflect current state

Table 5.3.1: Transformation characteristics

decision support tables must match that of the provided qualification output, with the exceptions of specific fields, like surrogate keys, and precision of calculations. The same technique has been successfully applied to other benchmarks, such as TPC-H and TPC-DS.

To assure that the transformations are defined within a DI tool, the specification defines the minimum requirements the data integration system must meet. These common characteristics of DI tools are specified at a high level, for example, the ability to read and write data to and from more than one data store and provide data transformation capabilities. In addition, the specification requires the DI system to

DSS Table	Characteristics of Transformations																
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
$T_{H,DimAccount}$	✓	✓	✓		✓		✓	✓									
$T_{H,DimBroker}$				✓	✓												
$T_{H,DimCompany}$	✓	✓	✓	✓	✓	✓	✓	✓									
$T_{H,DimCustumer}$	✓	✓	✓	✓	✓	✓		✓		✓			✓				
$T_{H,DimDate}$					✓												
$T_{H,DimSecurity}$		✓			✓	✓	✓				✓	✓	✓		✓	✓	
$T_{H,DimTime}$					✓												
$T_{H,FactTrade}$			✓	✓					✓		✓	✓					✓
$T_{H,FactCashBalances}$				✓							✓						✓
$T_{H,FactHolding}$				✓					✓		✓	✓		✓			✓
$T_{H,FactMarketHistory}$			✓	✓						✓	✓		✓				✓
$T_{H,FactWatches}$				✓		✓		✓			✓			✓			✓
$T_{H,Industry}$				✓													
$T_{H,Financial}$						✓	✓				✓			✓		✓	
$T_{H,FactProspect}$	✓	✓															✓
$T_{H,StatusType}$				✓													
$T_{H,TaxRate}$				✓													
$T_{H,TradeType}$				✓													

Table 5.3.2: Transformations and their characteristics

translate a DI specification into a DI application.

The order in which transformations are executed is left to the benchmark sponsor provided that all functional dependencies between tables of the decision support system are honored. This means when a dependent table column refers to a column in a source table, any rows in the source table that would change the outcome of processing a row in the dependent table must be processed before the dependent row. For instance, DimCustomer is fully processed before DimAccount because the account records refer to customer records. The specification defines these dependencies precisely.

The benchmark requires that at the end of each phase, all transformations must have completed successfully and their output data must be committed into the decision support system. Starting from the first incremental update phase, the decision support system must be operational and accessible to any user of the DI system.This implies that data that has been committed must remain visible to any other user.

5.3.1 History Keeping Dimensions

History keeping dimension tables retain information about changes to its data over time, while also allowing easy querying of current information. This is accomplished using both the primary key of the source system table, which is constant over time, and a surrogate key that is updated for each recorded change plus two additional fields, *EndDate* and *IsCurrent*. While EndDate would be sufficient to identify the most current record, in practice an additional field IsCurrent is added to simplifiy query writing. The EndDate of the *current* record is set when updated information is received, which essentially expires it. When querying a dimension to find the valid record for a given time, a condition like EffectiveDate $\leq$ *my_time* $<$ EndDate could be used. Using a NULL value for EndDate complicates these sorts of queries as these conditions will be UNKNOWN on current records, so additional logic would need to be added to account for that. To avoid this complication, a date far off into the future is used as the EndDate for current records, which allows a basic date range search to work for all records. Fact tables that reference a history keeping dimension include a foreign key reference to the surrogate key, not the natural key. The concept of a history keeping dimension is common in the industry, and is sometimes referred to as a *type 2 changing dimension* or a *type 2 slowly changing dimension*. Any transformation that inserts data into a history keeping dimension must execute one of the following two steps. When a record with a business key K does not exist in the dimension table the following transformations are performed:

- A unique surrogate key value must be assigned and included in the inserted record, that is, a dense sequence number.
- IsCurrent is set to TRUE to indicate that this is the current record corresponding to the natural key.
- The EffectiveDate field is set to a value specified by the transformation, or Batch Date if no value is specified.
- The EndDate field is set to December 31, 9999.

When a record with a business key K already exists in the dimension table the following transformations are performed:

1. Update the existing dimension table record for that natural key where IsCurrent is set to TRUE (these updates are known as 'Expiring' the record):
   (a) The current indicator field, IsCurrent, is set to FALSE to indicate that this is no longer the current record corresponding to the natural key.
   (b) The EndDate field is set to the EffectiveDate of the new record.
2. After expiring the existing record in the dimension table, a new record is inserted into the dimension table following the same transformation steps as those for inserting a new record above.

5.3.2 Example: DimAccount Transformations

Two of the more complex transformations are specified for the DimAccount table. The transformation for the historical load is different from the transformation for the incremental loads as data for the historical load is obtained from the CustomerMgmt.xml file while data for the incremental loads is obtained from the account.txt file. In this section we discuss the transformation for the historical load. For a description of the transformation in pseudo code see Algorithm 1. We refer to specific data elements in the XML document using XPath notation[8]. All references are relative to the context of the associated Action (/Action) data element.

Customer/Account/@CA_ID[9] is the natural key for the account data. New accounts may have missing information in which case the DI process has to insert a NULL value in DimAccount. Updated account information contains only partial data, that is, all properties that are missing values retain their current values in the DimAccount. All changes to DimAccount are implemented in a history-tracking manner.

When processing data from the XML-file we have to differentiate between the six different actions associated with customers and accounts, that is, *New* (new customer), *AddAcct* (add a new account), *UpdAcct* (update an existing account), *UpdCust* (update an existing customer), *CloseAcct* (close an existing account),*Inact* (inactivate an existing customer). For new accounts a new record with information from *AccountID*, *AccountDesc* and *TaxStatus* are filled with the corresponding XML elements. *Status* is set to 'ACTIVE'. *SK_Broker_ID* and *SK_Customer_ID* are set by obtaining the associated surrogate keys by matching Customer/Account/CA_B_ID with DimBroker.BrokerID and Customer/@C_ID with DimCustomer.CustomerID where the date portion of ./@$ActionTS$ >= EffectiveDate and the date portion of ./@$ActionTS$ <= EndDate. The *BrokerID* and *CustomerID* matches are guaranteed to succeed. In case where updates to an existing account are received, fields that exist in the source data are transformed to the target fields as done for new accounts. Fields that do not exist in the source data retain their values from the current record in DimAccount. For accounts that are closed *Status* is set to 'INACTIVE'. In

[8]`http://www.w3.org/TR/xpath`
[9]refers to the CA_ID of the account for customer

```
if @ActionType=NEW or ADDACCT then
    AccountID ← Customer/Account/@CA_ID;
    AccountDesc ← Customer/Account/@CA_NAME;
    TaxStatus ← Customer/Account/@CA_TAX_ST;
    SK_BROKER_ID ←
        SELECT BrokerID
        FROM DimAccount,DimBroker
        WHERE Customer/Account/@CA_D_ID = DimBroker.BrokerID ;
    SK_CUSTOMER_ID ←
        SELECT C_ID
        FROM DimAccount,DimCustomer
        WHERE Customer/Account/@C_ID= DimCustomer.CustomerID
        AND @ActionTS BETWEEN EffectiveDate AND EndDate ;
    Status ← ACTIVE;
else if @ActionType=UPDACCT then
    foreach source field with data do
        the same as for NEW and ADDACCT;
    end
    foreach source field without data do
        retain current values;
    end
else if @ActionType=CLOSEACCT then
    Status ← INACTIVE;
else if @ActionType=UPDCUST then
    foreach account held by customer do
        SK_CustomerID ← updated customer record;
    end
else if @ActionType=INACT then
    SK_CustomerID ← updated customer record;
    Status ← INACTIVE;
    foreach account held by customer do
        SK_CustomerID ← updated customer record;
        Status ← INACTIVE;
        IsCurrent, EffectiveDate, and EndDate ← according to algorithm to
          history keeping dimensions
    end
```

Algorithm 1: DimAccount historical load transformation

addition, a transformation rule is defined that requires that changes to a Customer also result in an update to all associated account records. These are implied changes to the account, that is, there is nothing in the source data that specifies which accounts must be updated. It is up to the implementation to identify the correct accounts and perform the required transformations. When an associated customer is updated, the SK_CustomerID field must be updated to the new customer surrogate key. In addition if an associated customer is set to inactive, the account must also be set to inactive. All of these changes to the account table must be handled as history keeping changes.

5.4 Execution Rules

TPC-DI benchmark models the two most important workloads of any mature DI system, one variant performs a historical load at times when the decision support system is initially created or when it is recreated from historical records, for example decision support system restructuring. The second variant performs periodic incremental updates, representing the trickling of new data into an existing decision support system. These two phases have very different performance characteristics and impose different requirement to the decision support system as it does not need to be queryable during the historical load, but it does need to be queryable during each incremental load. There are many different rates at which incremental updates may occur, from rarely to near real-time. Daily updates are common, and are the model for the TPC-DI benchmark. The combination of these two workloads constitutes the core competencies of any DI system. The TPC has carefully evaluated the TPC-DI workload to provide a robust, rigorous, and complete means for the evaluation of systems meant to provide that competency.

TPC-DI's execution model consists of the following timed and untimed parts. It is not permitted to begin processing of a phase until the previous phase has completed: (i) Initialization Phase (untimed), (ii) Historical Load Phase (timed), (iii) Incremental Update 1 Phase (timed), (iv) Incremental Update 2 Phase (timed), and (v) Automated Audit Phase (untimed).

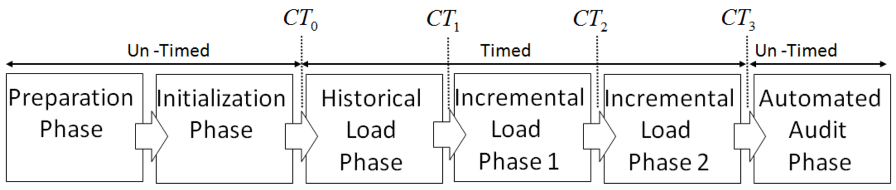


Figure 5.4.1: Execution phases and metric

The preparation phase contains setting up the system, installing all necessary software components and setting up the staging area. Before starting a measurement run the test sponsor chooses a scale factor that result in an elapsed time of each incremental update phase of 3600 seconds or less.

5.5 Metric

TPC-DI defines one primary performance metric and one primary price-performance metric. The performance metric is a throughput metric. It represents the number of rows processed per second as the geometric mean of the historical and the incremental loading phases. In order to calculate throughput numbers, we need to define the measurement interval and what we mean by rows processed. As indicated in Figure 5.4.1 TPC-DI defines four completion time stamps (CTs) to be taken at a precision of 0.1 second (rounded up), for example 0.01 is reported as 0.1. The number of rows processed in each phase is provided by TPC-DI's data generator, DIGen. The metric is then incrementally calculated as:

- CT_0: Complection of the Initialization
- CT_1: Completion of the Historical Load
- CT_i: Completion of Incremental Load $i \in \{1, 2\}$
- R_H: Rows loaded during the Historical Load
- R_{Ii}: Rows loaded during Incremental Load $i \in \{1, 2\}$

$$E_H = CT_1 - CT_0; E_{Ii} = CT_{(i+1)} - CT_i; i \in \{1, 2\} \tag{5.5.1}$$

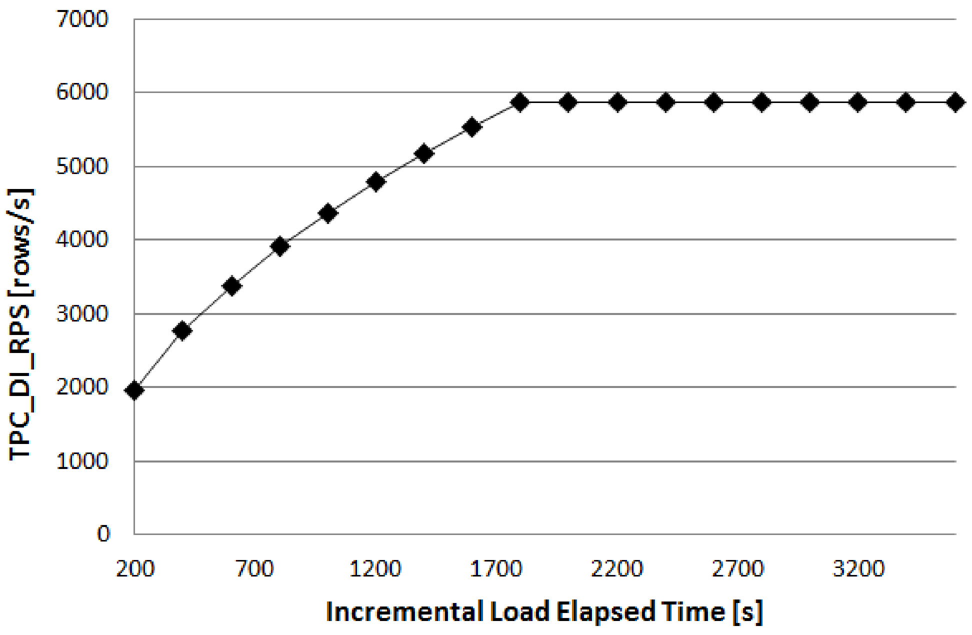


Figure 5.5.1: Scaling with incremental load time

$$T_H = \frac{R_H}{E_H}; T_{Ii} = \frac{R_{Ii}}{max(T_{Ei}, 1800)}; i \in \{1, 2\} \tag{5.5.2}$$

$$TPC\_DI\_RPS = \lfloor(\sqrt{T_H, min(T_{I1}, T_{I2})})\rfloor \tag{5.5.3}$$

Defining the elapsed time of a phase between the completion time stamps (CT) of its preceeding phase and it's own CT assures that all work defined in the benchmark is timed. The execution rules of TPC-DI define the historical and two incremental loads as functionally dependent, that is, work done in one phase has to be completed before the succeeding phase can start. Defining the start time of a phase as the completion time of its preceeding phase assures that all work that should be attributed to that phase is indeed timed. For instance, if the historical load phase reports all rows loaded, but indexes are still being maintained, the following incremental update phase either waits until all indexes have been maintained or it starts without using them suffering performance as a consequence.

The metric encourages the processing of a sufficiently large amount of data during the execution of the benchmark. The actual amount of data depends on the system's performance. The higher the performance of a system the more data it needs to process. The definition of the incremental load throughputs (see equations 5.5.2) entices the benchmark sponsor to achieve elapsed times $T_{Ii}i \in \{1, 2\}$ close to 1800s. The benchmark rules allow elapsed times less than 1800s, however, with a negative impact on the reported performance number. This is due to the *max* function in the denominator of the throughput functions T_{I1}, T_{I2}. It calculates the number of rows processed per second by dividing the actual rows loaded by the elapsed time of the load, but by at least 1800. Assuming that a system is capable of delivering load performance linear with data size, that is, $T_{Ii}(1) = T_{Ii}$ then $T_{Ii}(SF) = n_{Ii} * SF * T_{Ii}, i \in \{1, 2\}, n_{Ii} > 1$, and $T_H(1) = T_H$ then $T_H(SF) = n_H * SF * T_H, n_H > 1$, then T_{I1} and T_{I2} increase linearly until an overall elapsed time of 1800s is achieved and stay flat thereafter. Figure 5.5.1 shows the flattening effect on the primary performance metric TPC_DI_RPS. On the x-axis its shows the elapsed time for the incremental load phase and on the y-axis it shows the main metric ($\frac{rows}{s}$). The graph shows that the metric increases until an elapsed time of 1800s is reached for the incremental load phase.

The metric entices good performance during both types of loads, historical and incremental. This is achieved by using the geometric mean to combine the historical and incremental throughputs into one meanigful number. Because by its definition the geometric mean treats small and large numbers equally. Hence, engineers are enticed to improve the performance of all phases of the benchmark regardless what their overall elapsed times are. This is especially important if there is a large elapsed time discrepancy between the historical load and incremental loads. For instance, reducing a the historical load from 100s to 90s, that is, 10% has the same effect on the final metric as if the incremental load with the smaller elapsed times is reduced from 10s to 9s. The above is not true when applied to "absolute" improvements.

The metric encourages a constant incremental load performance. Production systems execute many more than the two incremental load phases defined in TPC-DI. It would be prohibitive to mandate the execution of many incremental load phases as part of a benchmark due to time constraints. However, TPC-DI ensures that there is

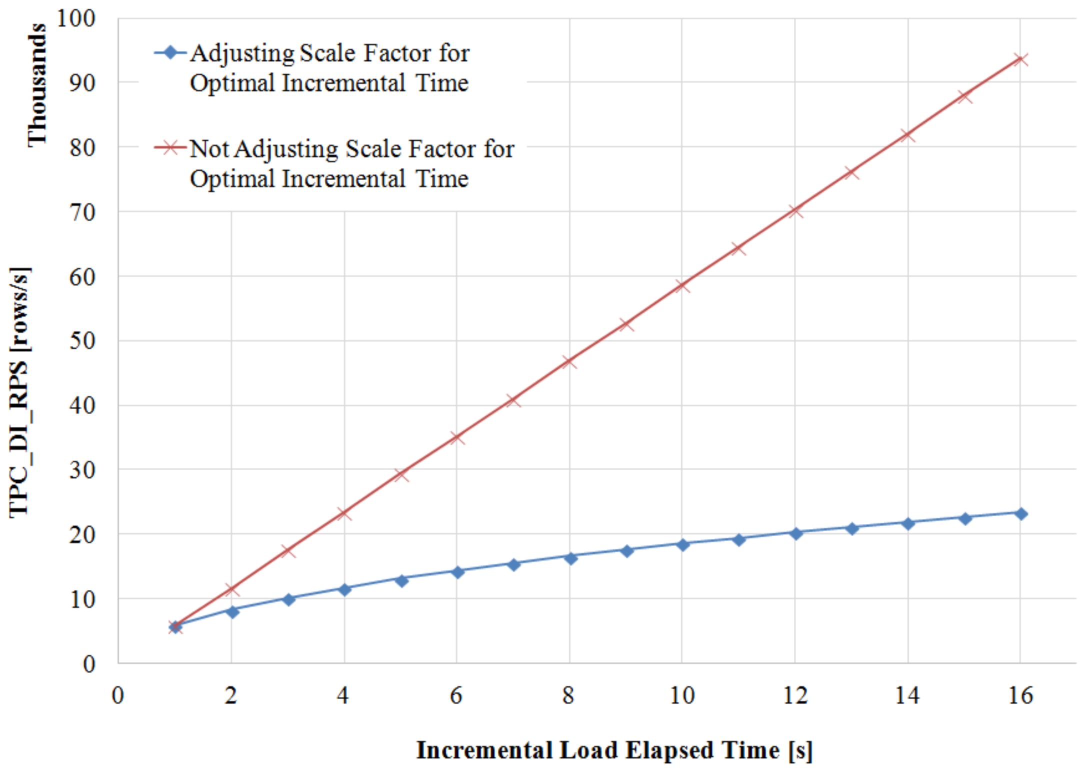


Figure 5.5.2: Linear scaling with adjusted elapsed time

no negative performance effect of executing multiple incremental load phases, that is, a slow down from one incremental load phase to the next by only including the lower of the two throughputs (see *min* in primary performance metric - Equation 5.5.3).

The metric scales linearly with system size. A very important feature of a performance benchmark metric is that it allows to showcase the scalability of a system (scale-out and scale-up), that is, a system with double the number of resources, for example sockets, cores, memory etc. should show double the performance in TPC-DI. However, this is only true if for each system size a scale factor is chosen that results in an incremental elapsed time of 1800s. Figure 5.5.2 shows that the primary performance metric increases linearly if the scale factor is adjusted for achieving an elapsed time of 1800s in the incremental load phase. On the x-axis it shows system size as number of cores and on the y-axis its shows the metric. The graph with the square labels shows the metric when increasing the number of cores and keeping the scale factor constant. In this case the scale factor was chosen such that an elapsed time of 1800s was achieved with 1 core. As the number of cores is increased and elapsed times decrease the overall performance increases only slighly. If, however, the scale factor is adjusted for the increase in system size, the primary performance metric increases linearly as indicated by the triangular graph.

5.6 Performance Study

As of this writing, there are no published results of TPC-DI. However, implementations are being developed and have been used to evaluate various aspects of the benchmark. The following sections discuss performance related topics based on observations made from real implementations of the workload.

5.6.1 Scalability

In order for a benchmark to have longevity, it must provide a workload that can scale as hardware and software systems become increasingly powerful. An implementation of the workload may have bottlenecks or the system it is run on may have constraints that limit its scalability, but the definition of the workload must not contain any requirement that inherently prevents scaling of implementations. Workload requirements that force implementations into bottlenecks can create situations where hardware and software components can not be adequately utilized during a benchmark run.

Using an implementation from IBM, the TPC-DI workload was run on the same system with linearly increasing scale factors. Figure 5.6.1 shows the results. The x-axis shows the normalized Source Data Set size, and the y-axis shows the normalized time. Using the normalized numbers, it is easy to see as the source data set increases by a factor of 2, the elapsed time increases correspondingly.

This demonstrates that the implementation is scaling up as expected. It was also

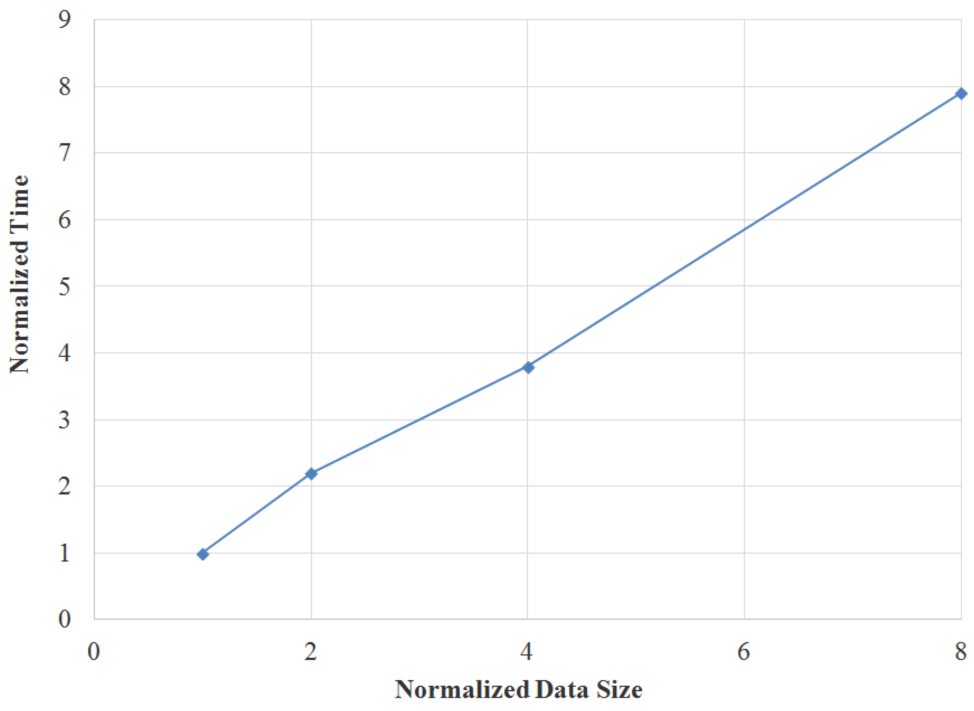


Figure 5.6.1: Total lapsed time scaling

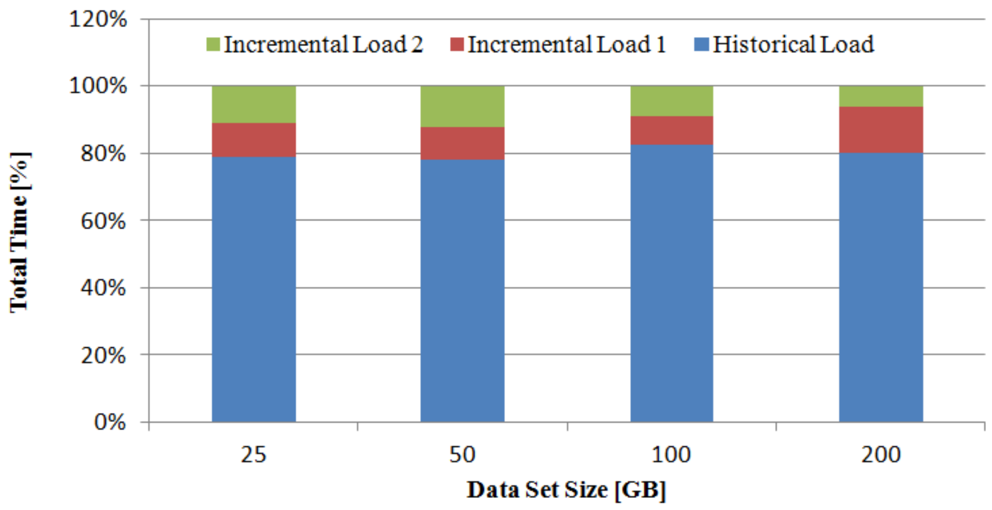


Figure 5.6.2: Relative phase time

observed that when the hardware resources were scaled to match the data set scaling, the execution time remained flat. This indicates the implementation is able to scale out to utilize available hardware resources. While these results are specific to this implementation, it indicates that the workload definition itself is scalable, that is, it allows for scalable implementations to be created.

5.6.2 Estimating Benchmark Execution time

The TPC-DI workload consists of 3 measured phases, the historical load and two incremental updates. The historical load phase has a set of required transformations, and processes more and larger files. There is no time limit for this phase. The transformations required for each incremental update phase are identical, and the input data sets are different but similar in size. When planning for a benchmark run, there may be a question as to how long it will take for the benchmark run to complete.

Figure 5.6.2 shows the relative amount of time spent processing each phase, for a specific implementation at 4 different scale factors. The historical load phase fairly consistently represented 80% of the overall time running the benchmark, while each

Throughput	**Time**	**Weighted** $(Th * t)$
10000	3000	30,000,000
17000	18000	306,000,000
2000	2700	5,400,000
8000	4200	33,600,000
0	900	0

Table 5.6.1: Throughput metric example

incremental update made up about 10%. This information can be used to project how long it will take for a full benchmark to complete from a small sample run. Because the incremental update phases are going to run somewhere between 30 and 60 mins and will be 20% of the overall runtime, the expectation would be that execution time for a full volume benchmark run would be between 5 and 10 hours for this implementation. This can be expressed in a simple formula, $1800/p < et < 3600/p$, where et is the expected time (in seconds) and p is the proportion of time spent in an incremental update phase. While the portion of time spent in the phases may not be match this specific implementation, any implementation that is scaling well can use to method to estimate the time for a full volume run.

5.6.3 Phase Throughput

Each of the 3 benchmark phases is made up of a set of transformations, and a validation query that executes at the end of the batch. So, although the throughput for each phase is calculated a single number (total rows/total time), the actual throughput of the system at any given point during the run may vary greatly from the final calculated throughput. In fact, within a given phase the calculated throughput could be considered to be the average of the achieved system throughputs, weighted by the amount of time the system was sustaining each throughput. For example, assume the historical load has 375 million records to process and completes in 8 hours (28800s), with the breakdown of the significant throughputs given in the chart below. The calculated throughput of the phase would be 13020.8 TPC_DI_RPS.

The weighted average of the throughputs yields $\frac{375000000}{28800} = 13020.8$. This demonstrates an important characteristic of the metric. While higher throughputs are rewarded and lower throughputs have a negative impact, short-lived spikes of either kind are not significant. The throughputs that are sustained for the longest periods of time have the greatest impact. The segment with throughput 0 represents the batch validation query that is executed at the end of each phase. In terms of throughput, the time spent in this segment is pure overhead, that is, it is not possible to process any rows during this phase so the time spent can only lower the overall throughput. Therefore, it is in the test sponsor's interest to configure the data warehouse such that this query performs as efficiently as possible. This is especially important for the incremental update phases. Test runs have shown that the execution time of the batch validation query remains fairly constant between the measured phases. While a 900 second run time is relatively small for a 28800 second historical load, incremental update phases have a maximum time of 3600 seconds, so a 900 second run time would 25% of the overall time or more for the phase.

Also, the incentive for an incremental update to run at least 30 mins (1800 seconds) can be understood using a similar analysis. Since the minimum amount of time that can be reported is 1800 seconds, a run that ends in N seconds less than 1800, effectively has 0 throughput sustained for N seconds. As the difference from 1800 gets larger, the more significance the 0 throughput has to the calculated throughput.

Chapter 6

TPC-DS V2: First Industry Standard Benchmark for Measuring the Performance of SQL-Based Big Data Systems

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 [87], MapReduce [88], Pig [89], 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, for example, Hive [90], Stinger [91], Impala [92], Shark [93], Presto [94], and Spark [58]. Many RDBMS vendors are following suit by adding Hadoop support into their SQL engines, for example, IBM's Big SQL [95], Oracle's Big Data SQL [96], and SAP's Vora [97].

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, that is, 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 [15, 16, 17, 18]. 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 remainder of this chapter is organized in two sections. Section 6.1 develops the changes introduced in TPC-DS V2, that allow SQL-based big data solutions to run and publish benchmark results. Motivation and analysis is provided for major changes. Section 6.2 provides experimental results of four hardware and software combinations that are typically deployed in the big data use case TPC-DS V2 aims to address, including a detailed analysis of the experimental results.

6.1 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 [98] for the complete specification).

6.1.1 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 of techniques and algorithms to implement performance enhancements and to enforce data correctness that both 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 6.1.5).

6.1.2 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), that is, 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, for example, 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.

6.1.3 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, that is, maintenance of history and non-history keeping dimensions, inserts into fact tables, and deletes from fact tables. 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 that are currently deployed in big data implementations of DSS. The insert and delete operations logically delete old facts, for example, old sales transactions to make room for new fact data, that is, 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.

6.1.4 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 are modified. Most of the modifications address inconsistencies between the functional

Table 6.1.1: Query modifications applied to V2

Queries	**Modification**
10, 35	Rewrote disjunctions of exist predicates into exist predicates of unions, for example `exists (SubQuery1) OR exists (SubQuery2)` into `exists (SubQuery1 UNION ALL SubQuery2)`
34, 56, 64, 73, 75, 76	Added additional columns to the order-by clause to make the 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. Changed `coalesce(ws_qty,0)>0 and coalesce(cs_qty, 0)>0` to `(coalesce(ws_qty,0)>0 or coalesce(cs_qty, 0)>0)`. Changed `(coalesce(ws_qty+cs_qty,1)` to `(coalesce(ws_qty,0)+coalesce(cs_qty,0))`. 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*

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 6.1.1 lists the major query changes.

6.1.5 Metric and Execution Rules

The execution rules and metric in V2 have been redesigned to emphasize the performance characteristics of big data systems.

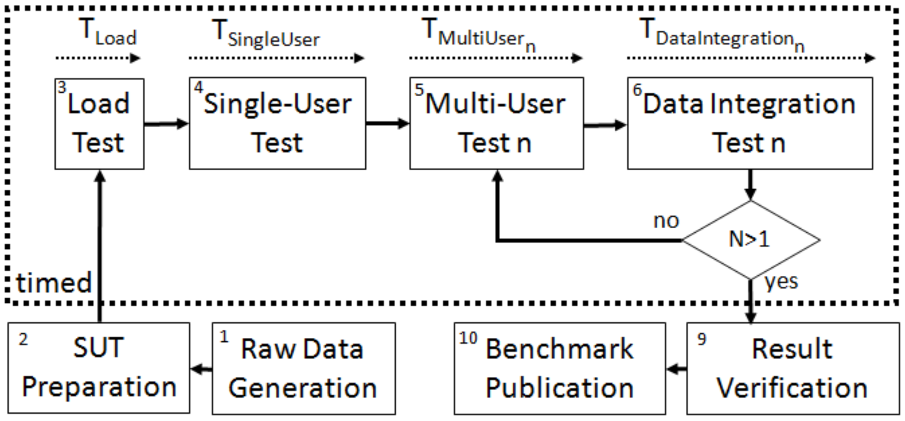


Figure 6.1.1: High level execution rules

Figure 6.1.1 illustrates the timed and the untimed phases of the execution of a TPC-DS V2 benchmark run. The generation of the raw data set, that is, 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 6.1.1) questions whether a load time is necessary at all. One could argue that systems that efficiently query external data, that is, 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

Table 6.1.2: 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

the notion of a *database location*, for example, 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 execute 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 system's 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 generally extremely difficult to police and, to some extent, 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 [31]. 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.

$$QphDS@SF = \left\lfloor \frac{SF * Q}{\sqrt[4]{T_{PT} * T_{TT} * T_{DI} * T_L}} \right\rfloor \tag{6.1.1}$$

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 that the elapsed time of the multi-user run becomes the dominant contributor to the metric for large S_q.

6.2 Experimental Results

We run our tests against four different setups, which resemble the diversity of systems being used to run DSS workloads. Because of the 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 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.

6.2.1 Data Scan Analysis

One of the main system factors of resource consumption in answering big data queries is 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, for example, partitioning and indexing, and whether the system supports vertical pruning of tables to eliminate unnecessary columns, for example, columnar access.

We establish an estimate for the upper bound of data needed to compute a query result by assuming that all tables referenced by the query are accessed fully, that is, no vertical or horizontal table pruning and disregarding common subexpression elimination of *with clauses* 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, for example, 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, that is, 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 follows:

$$\sum_{i=1}^{99} \sum_{t \in TQ_i} ARL(t) * C(t) \approx 260,000GB \tag{6.2.1}$$

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, we can estimate the lower bound for the amount of data read to answer all 99 queries as follows:

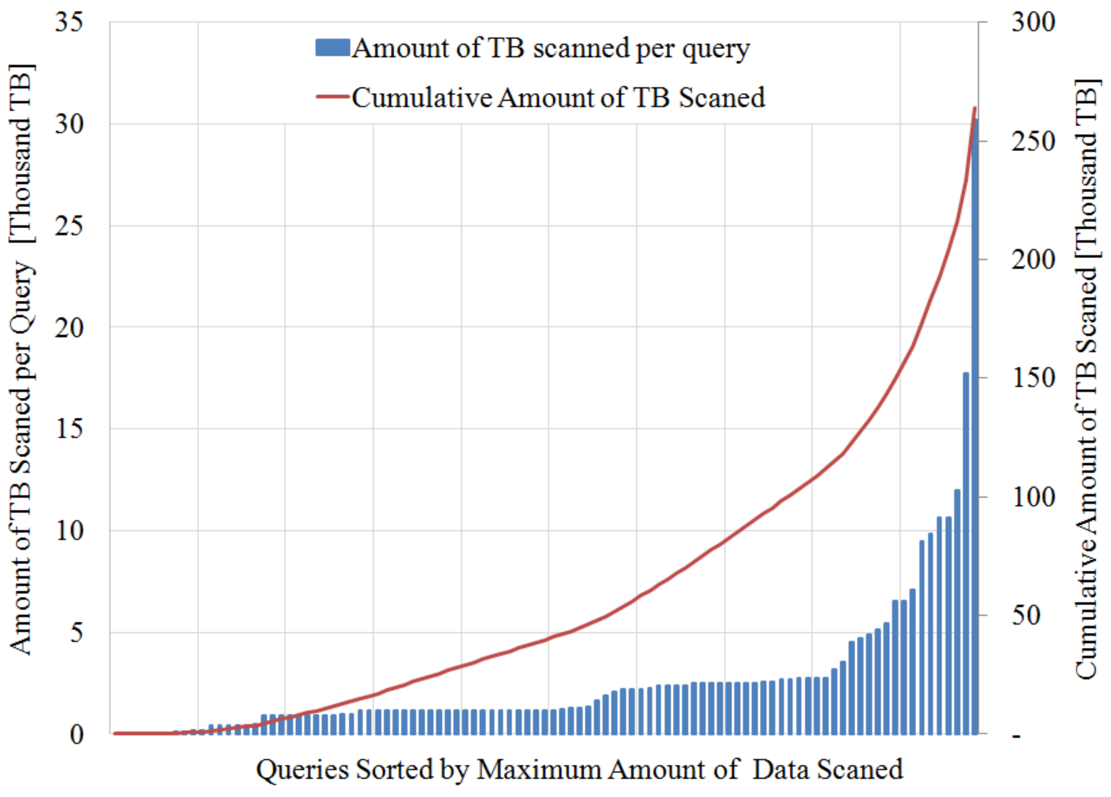


Figure 6.2.1: Upper bound of data access by queries

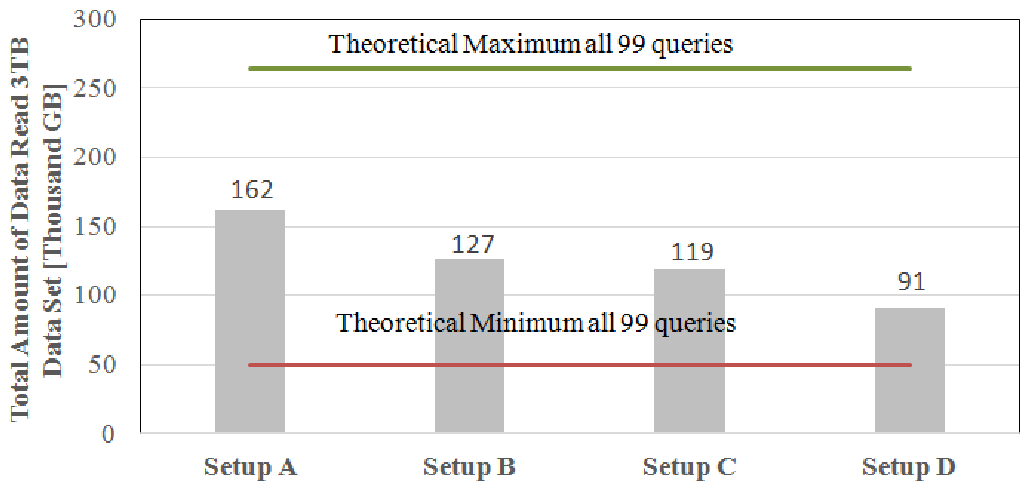


Figure 6.2.2: Minimum and maximum data access

$$\sum_{i=1}^{99} \sum_{t \in TQ_i} \sum_{c \in C_t} ACL(c) * C_{hp}(t) \approx 50,000GB \qquad (6.2.2)$$

Figure 6.2.1 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 6.2.2 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 systems reduce the amount of data read to the theoretical minimum of about

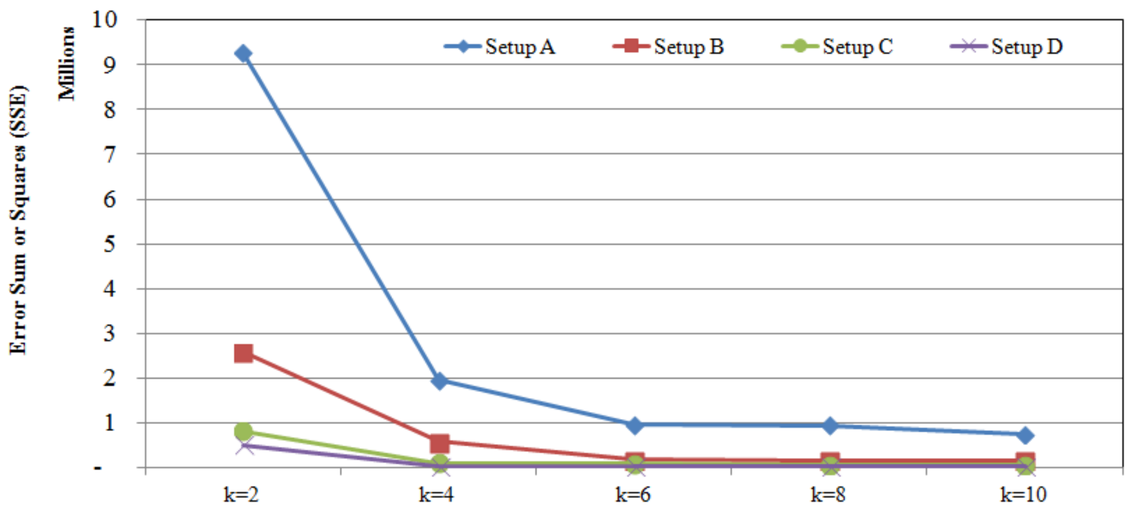


Figure 6.2.3: Sum of squared errors for each setup

50,000GB. We should also note that there is a very large discrepancy between the system reading the most and the system reading the least amount of data of about 70,000GB.

6.2.2 Single-User Test Analysis

The single-user test, aka 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 to see 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.

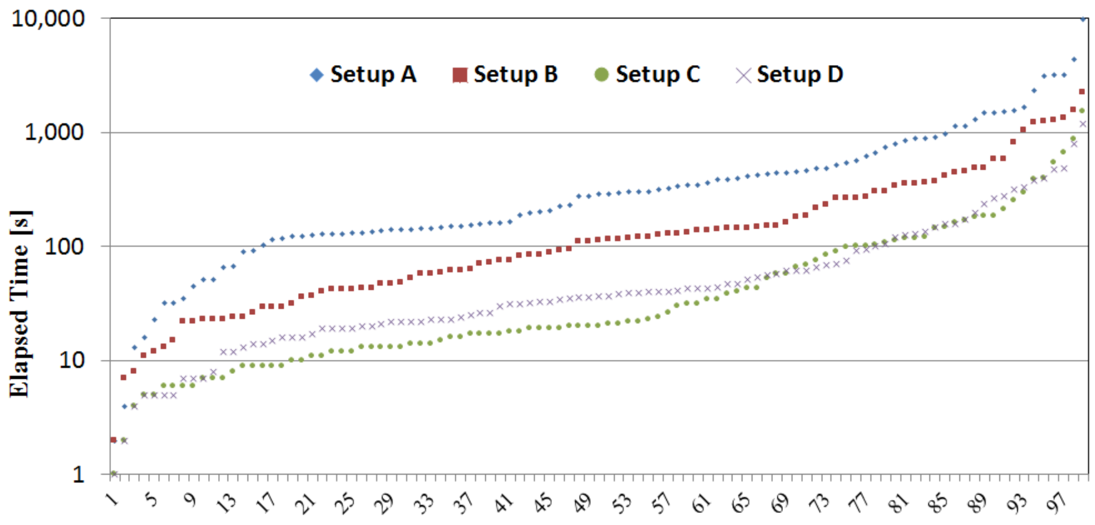


Figure 6.2.4: Sorted elapsed times

Figure 6.2.4 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 the following three methods:

1. Coefficient of Variation Analysis

2. K-means Cluster Analysis
3. Normalized Elapsed Time Analysis

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. Since 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 [99].

Intra Setup K-means Analysis

The coefficient of variation analysis gives us a good idea of 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 [100] 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 [101]. Our dataset size

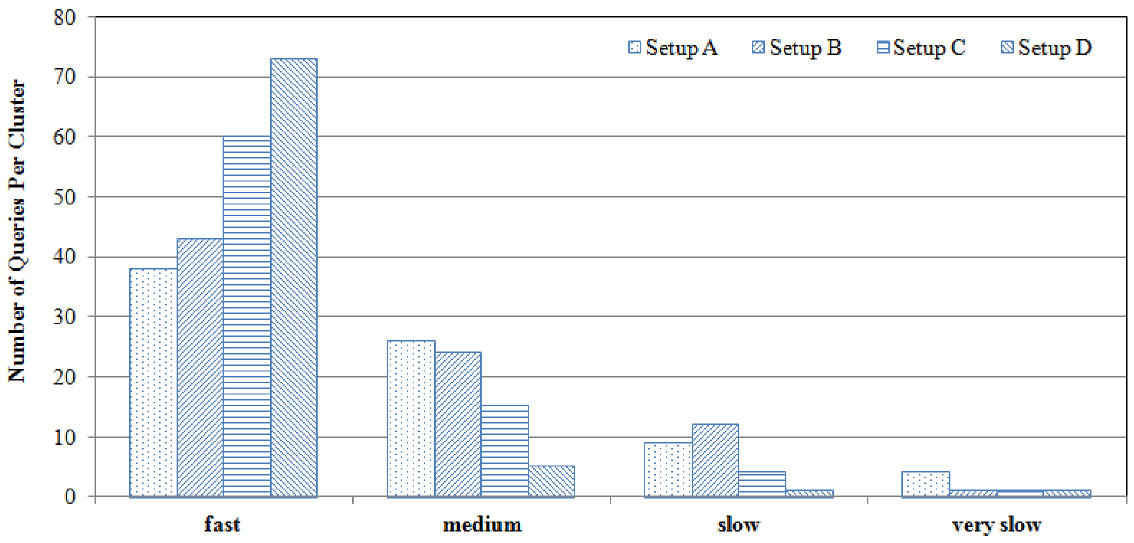


Figure 6.2.5: Number of queries in each cluster

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 follows:

$$\mu_{P,s,k} = \frac{1}{|Q_k|} \sum_{q \in Q_k} T_{P,s}(q) \tag{6.2.3}$$

We then calculate the SSEs as follows:

$$SSE_{P,s,k} = \sum_{k=1}^{k} \sum_{q \in Q_k} (\mu_{P,s,k} - T_{P,s}(q))^2 \tag{6.2.4}$$

Figure 6.2.3 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 6.2.5 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, 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 the slow category of Setup D that makes 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.

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 $T_{P,s}(q_i), 1 \leq i \leq 99$, we calculate the mean query elapsed time as follows:

$$\mu_s = \frac{1}{99}\sum_{i=1}^{99} T_{P,s}(q_i), 1 \leq i \leq 99 \tag{6.2.5}$$

We express the normalized response times for each query as:

$$T_{P,s}^{norm}(q_i) = \frac{T_{P,s}(q_i)}{\mu_s}, 1 \leq i \leq 99 \tag{6.2.6}$$

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 follows:

$$\mu(q_i) = \frac{1}{4}\sum_{s=1}^{4} T_{P,s}^{norm}(q_i) 1 \leq i \leq 99 \tag{6.2.7}$$

we calculate the coefficient of variation function as follows:

$$CV(q_i) = \sqrt{\frac{1}{99}\sum_{i=1}^{99}(T_{P,s}^{norm}(q_i) - \mu(q_i))^2}, 1 \leq i \leq 99 \tag{6.2.8}$$

The coefficient of variation function CV, plotted in Figure 6.2.6, shows the relative dispersion of the normalized elapsed times of the four setups on a ratio scale. The

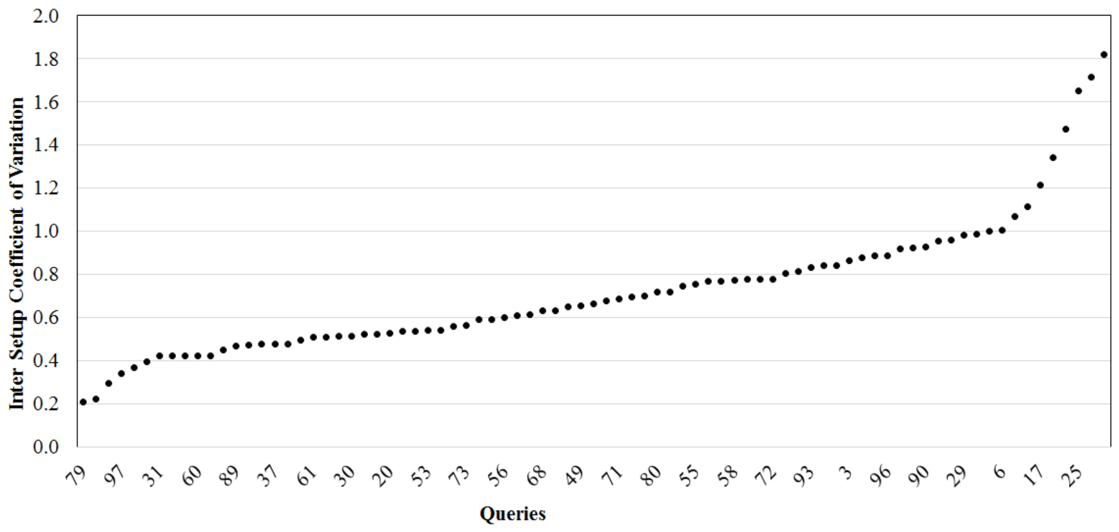


Figure 6.2.6: Single-user normalized elapsed times of all setups

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 6.2.1 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 Systems B and C result in a 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).

Table 6.2.1: Elapsed time & data scanned by queries with high normalized elapsed time dispersion

Query	Elapsed Time[s]				Data Scanned [GB]			
	A	B	C	D	A	B	C	D
12	51	7	4	185	276	4	54	532
16	1,508	22	5	11	2219	81	59	217
17	3,178	263	89	3	3,100	2,021	1,547	98
25	3,177	76	34	3	3,053	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	2,581	1,485	1,348	1,298	37,854

6.2.3 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, for example, IO and CPU. Next,

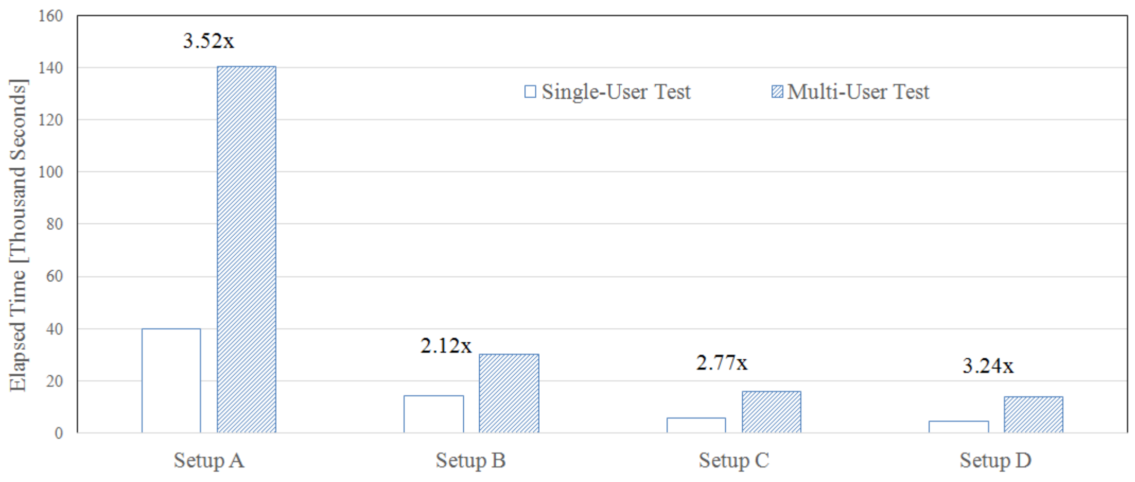


Figure 6.2.7: Comparison single-user/multi-user tests

we discuss the CPU and IO resource utilization over time.

Figure 6.2.8 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 6.2.2 of Section 6.2.1. 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 6.2.9. 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 6.2.9).

Figure 6.2.10 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

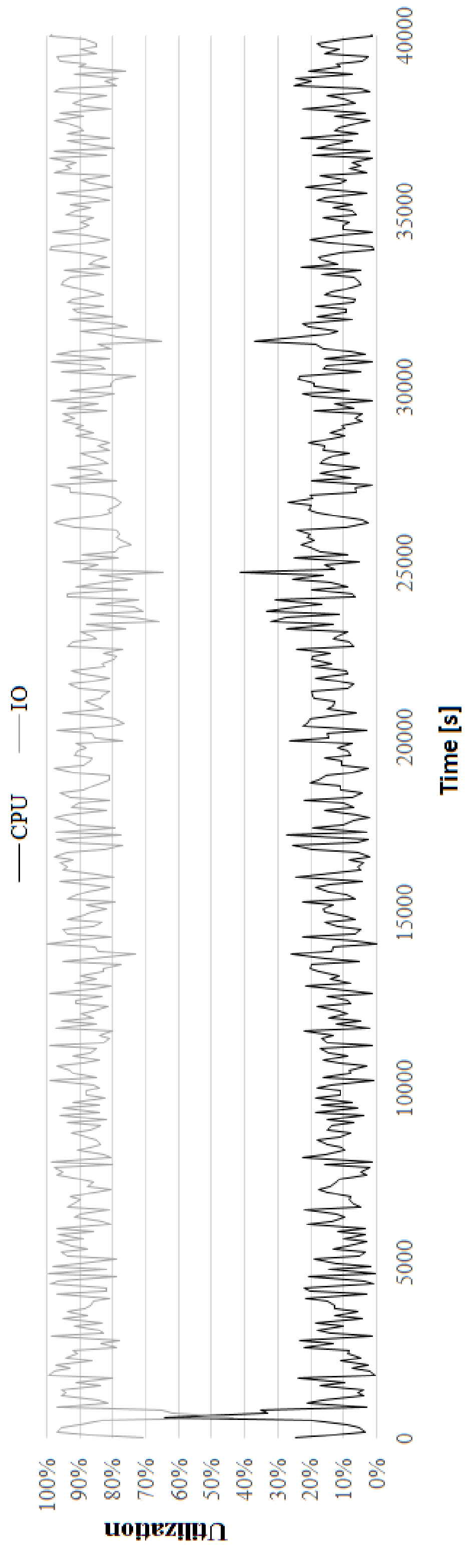


Figure 6.2.8: Setup A - CPU and IO utilization for the single-user test

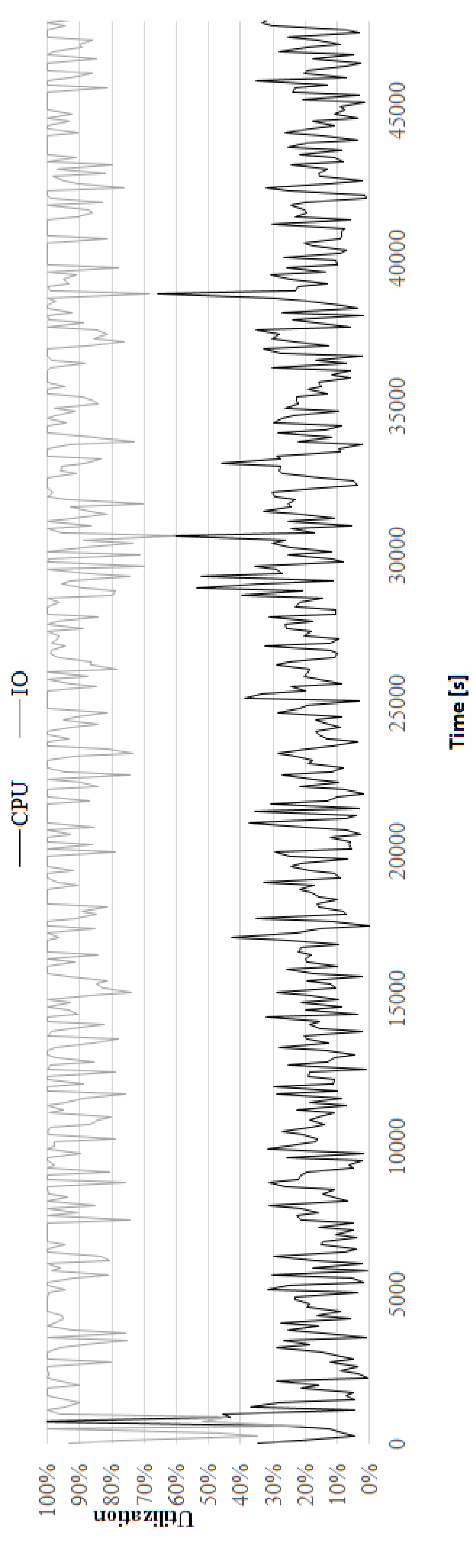


Figure 6.2.9: Setup A - CPU and IO utilization for the multi-user test

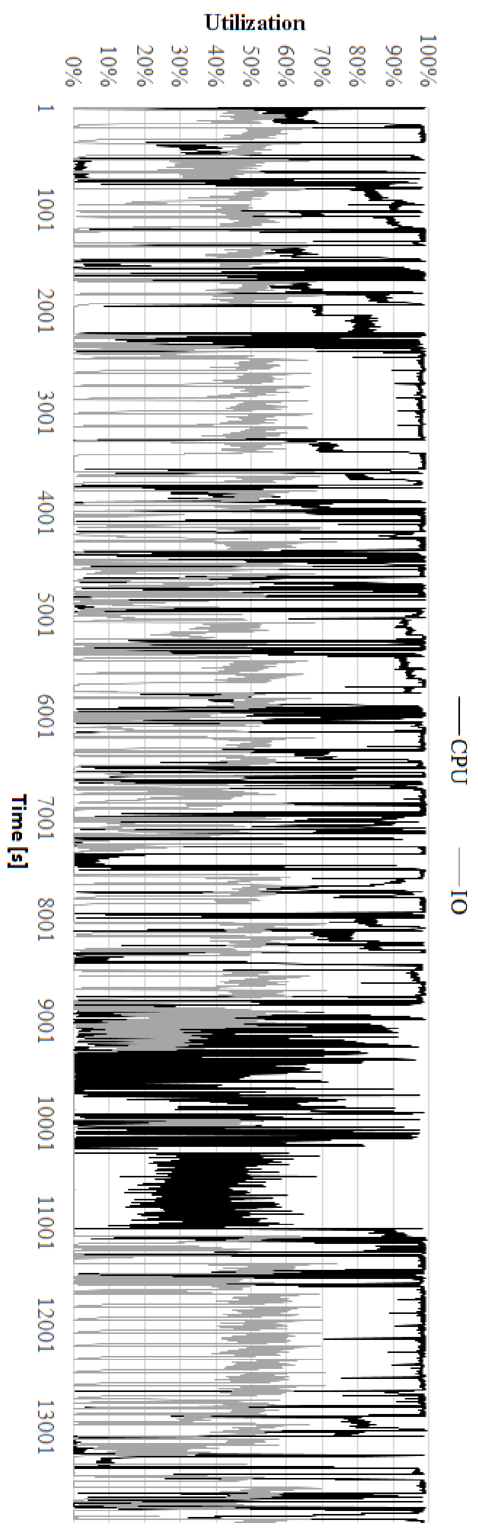


Figure 6.2.10: Setup B - CPU and IO utilization for the single-user test

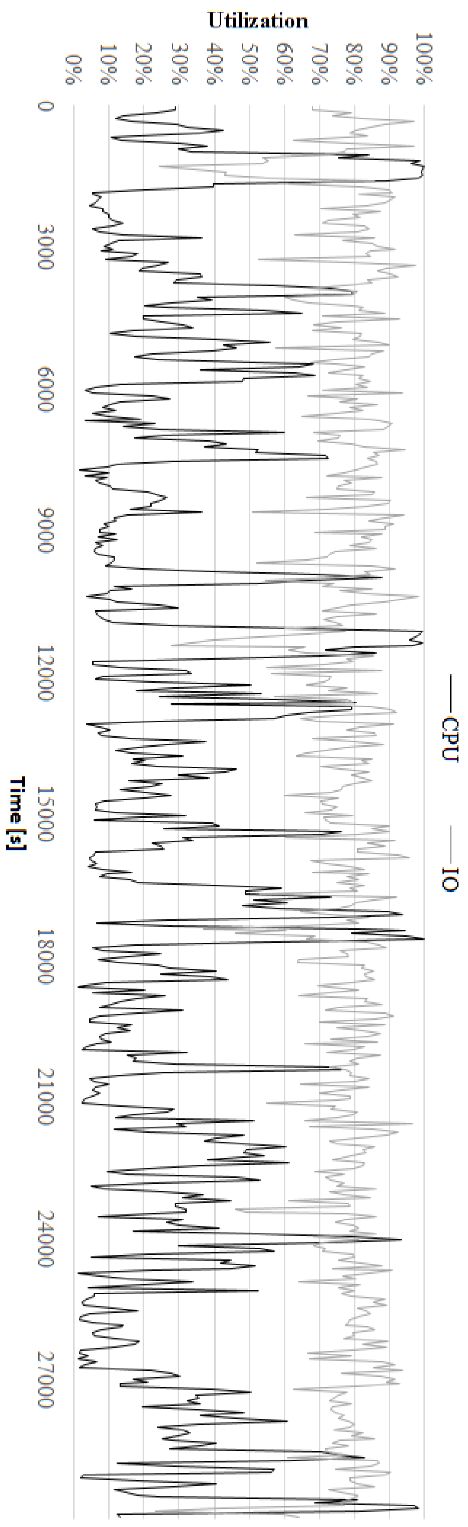


Figure 6.2.11: Setup B - CPU and IO utilization for the multi-user test

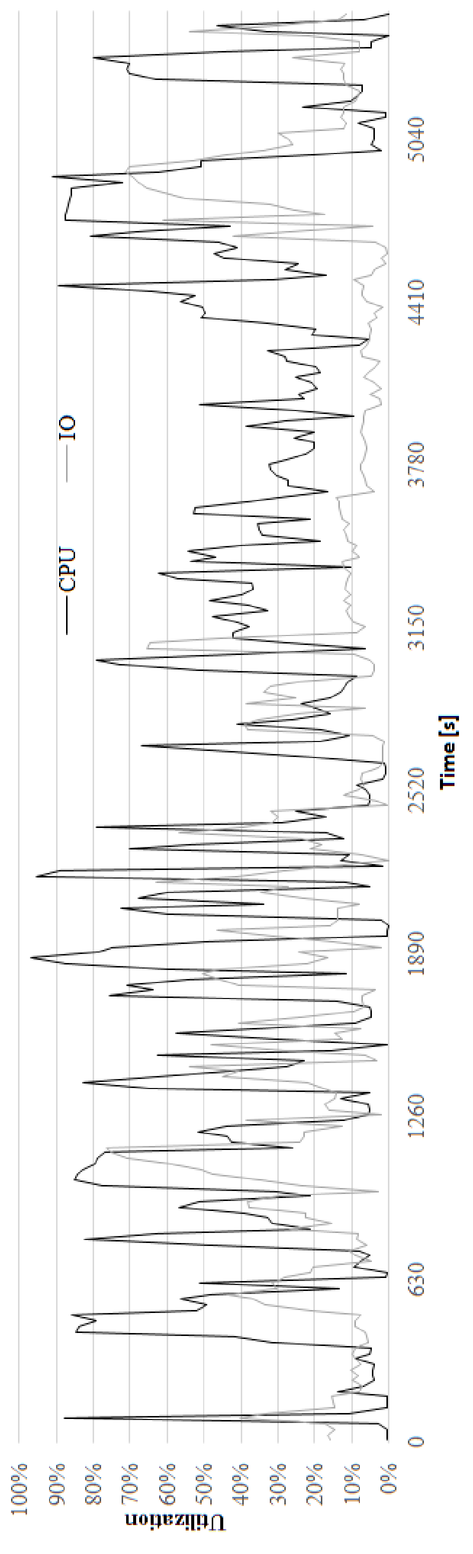


Figure 6.2.12: Setup C - CPU and IO utilization for the single-user test

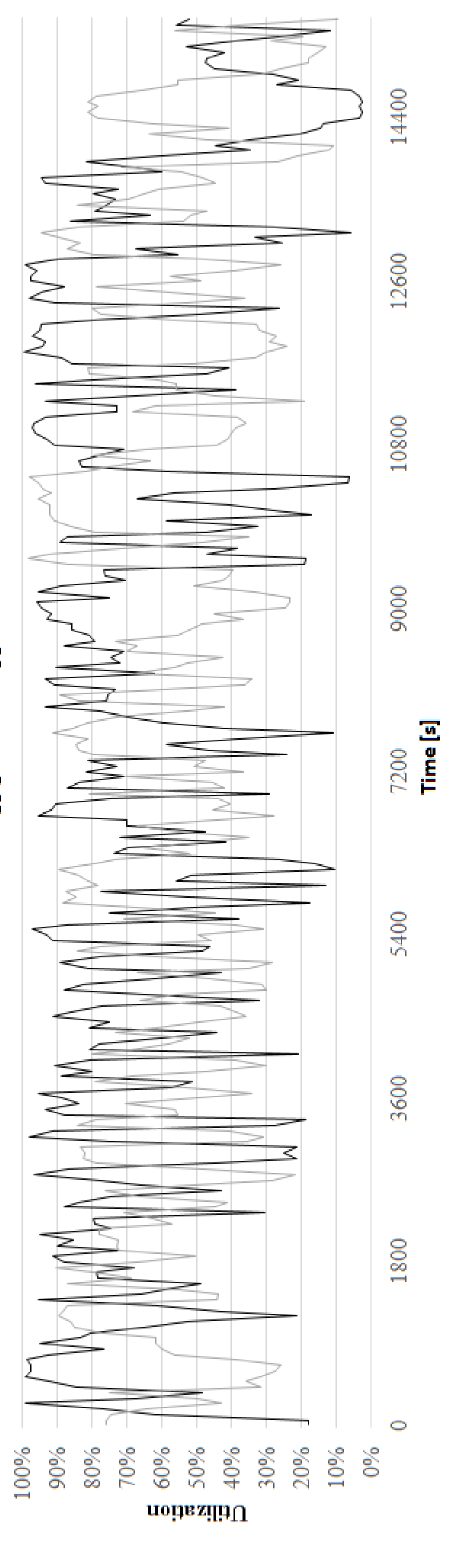


Figure 6.2.13: Setup C - CPU and IO utilization for the multi-user test

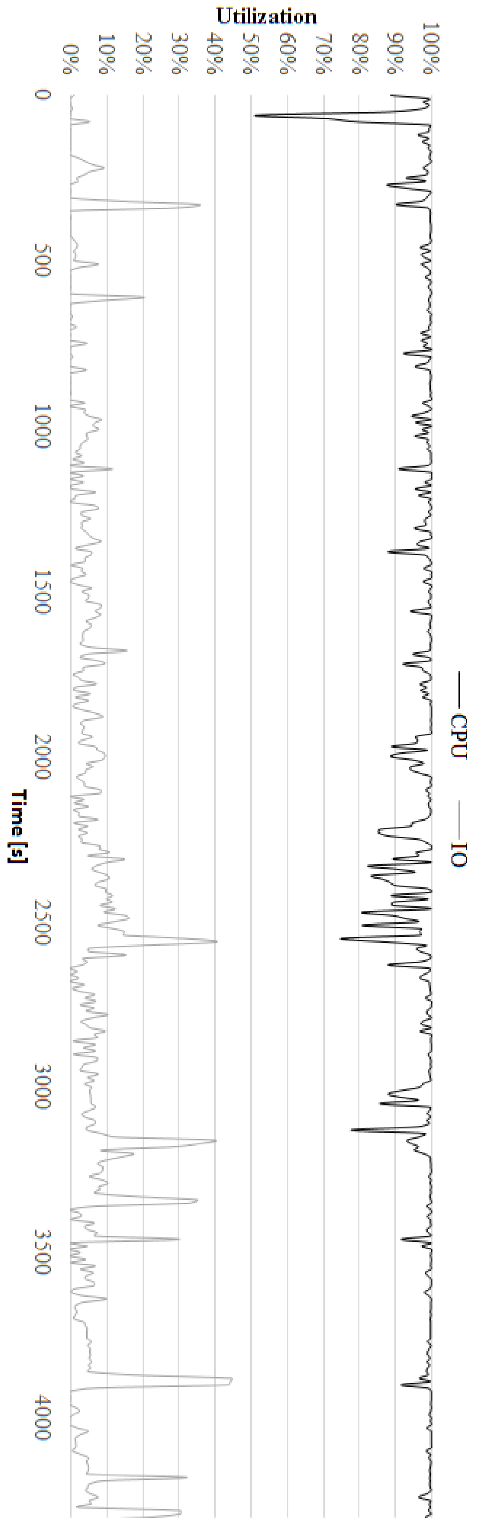


Figure 6.2.14: Setup D - CPU and IO utilization for the single-user test

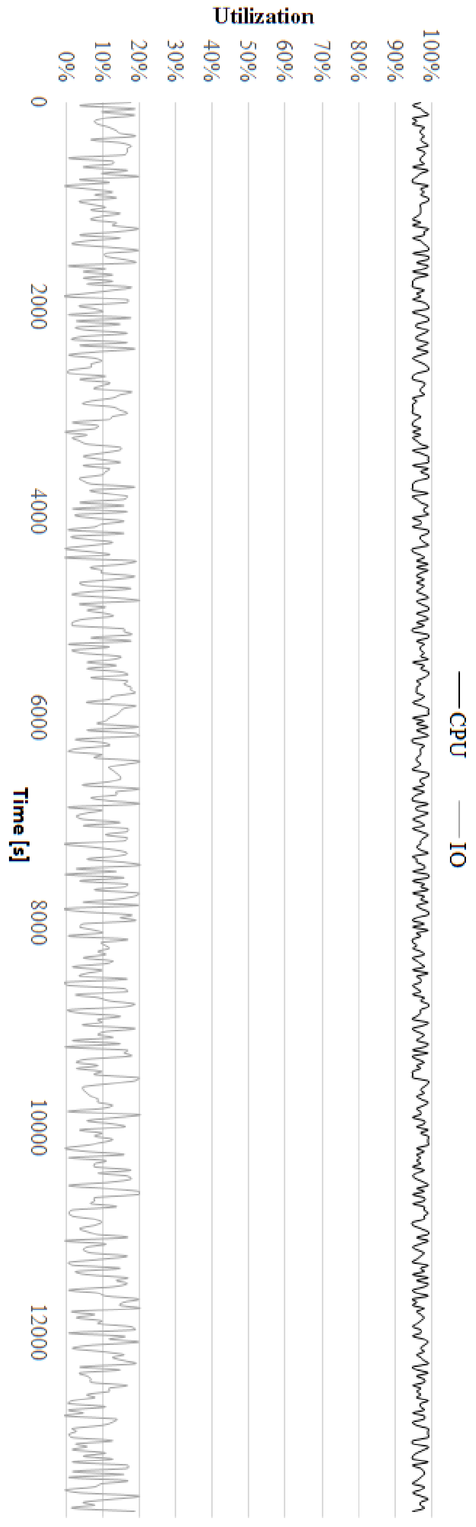


Figure 6.2.15: Setup D - CPU and IO utilization for the multi-user test

I/O capacity of the system. At no time interval is the system I/O bound.

Figure 6.2.11 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 it 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 6.2.12 and 6.2.13 show the corresponding resource utilizations for the single- and multi-user runs of Setup C. This setup behaves similar to Setup B as both the 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 6.2.14 and 6.2.15 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 a 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.

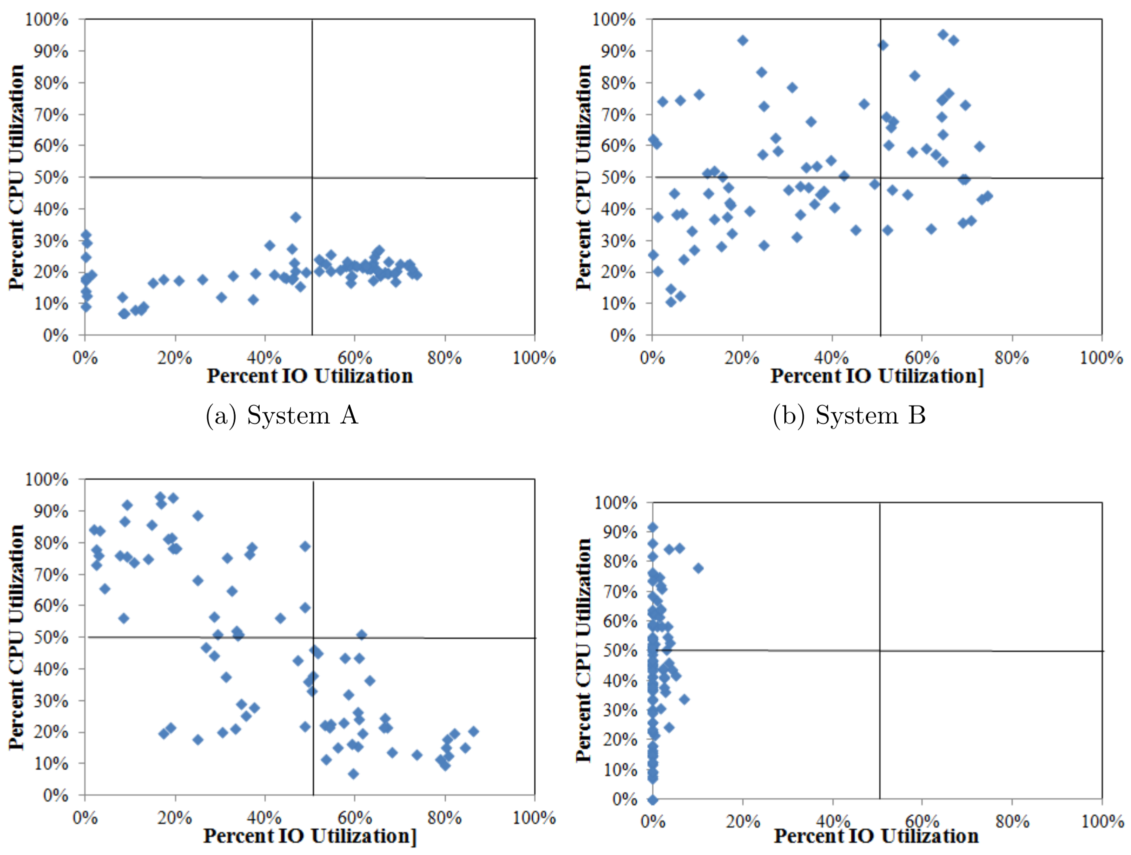
Comparing Single-User and Multi-User Tests

Figure 6.2.7 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 6.2.7. 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 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.

6.2.4 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. Analyzing systems that were used to publish TPC-H benchmark results, it becomes apparent that CPU, IO, inter-node communication, and memory are the most important system resources for executing decision support queries. 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 the following four resources: (i) CPU, (ii) I/O, (iii) memory, and (iv) network. The sets in Figures 6.2.16-6.2.18 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



(a) System A

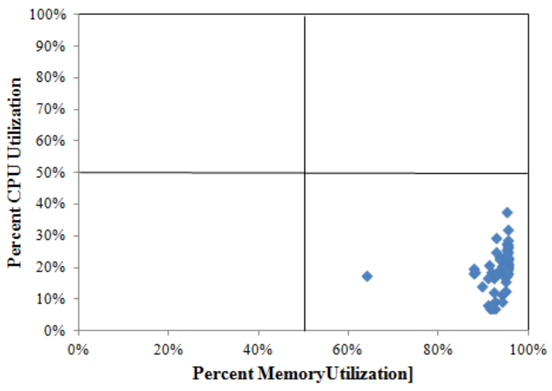
(b) System B

(c) System C

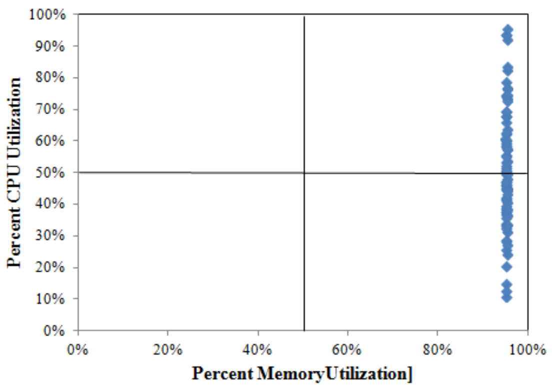
(d) System D

Figure 6.2.16: IO-CPU utilization

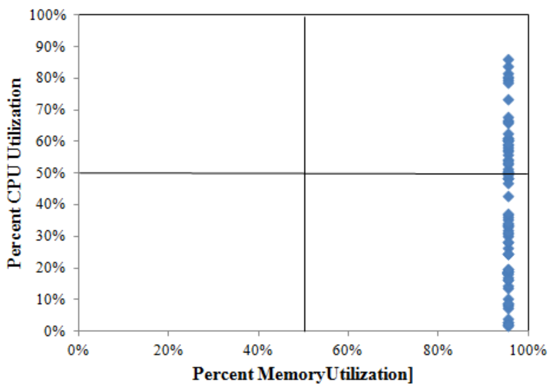
CPU is [*thread*], that is, hardware threads of a huperthreaded x86 processor, and includes user and system time as reported by tools, such as *vmstat*. The unit for IO is [*MBytes/s*] and includes all read and write operations to all disks attached to the system as reported by tools such as *iostat*. The unit for memory is [*GB*] and includes all physical memory allocated by all processes on the system including the file system cache. The 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



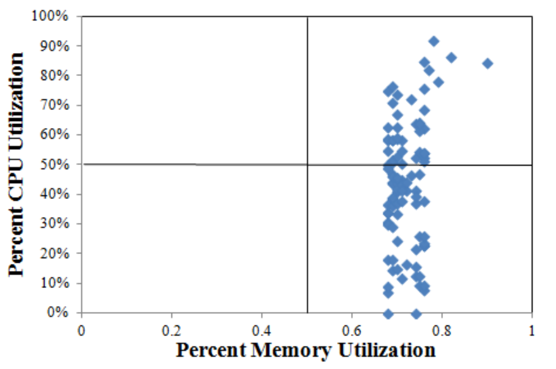
(a) System A



(b) System B



(c) System C



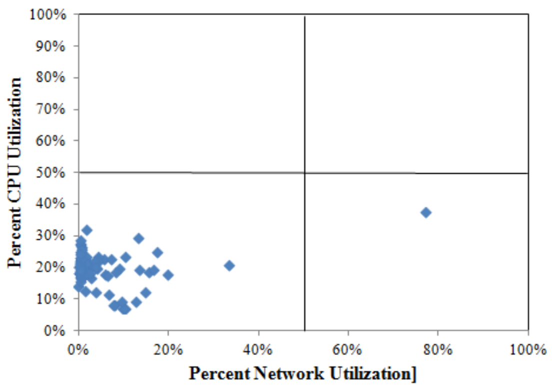
(d) System D

Figure 6.2.17: Memory-CPU utilization

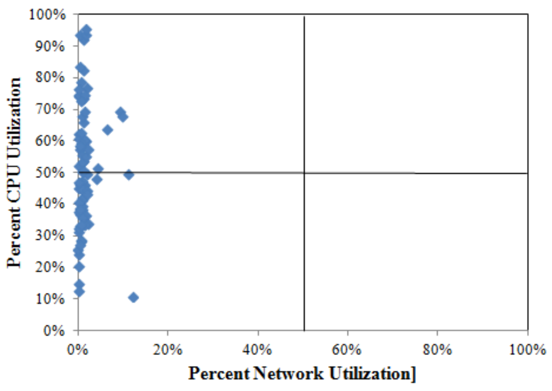
Resource r on System $s \in \{A, B, C\}$ for each Workload w, expressed in percent, can be computed as the sum of the ratios of actual resource utilization and maximum resource available during each Time Interval t.

$$U_{avg}(r, s, w) = \frac{100}{T(s, q)} \sum_{t=1}^{T(s,w)} \frac{U_{measured}(r, s, w, t)}{U_{max}(r, s)} \tag{6.2.9}$$

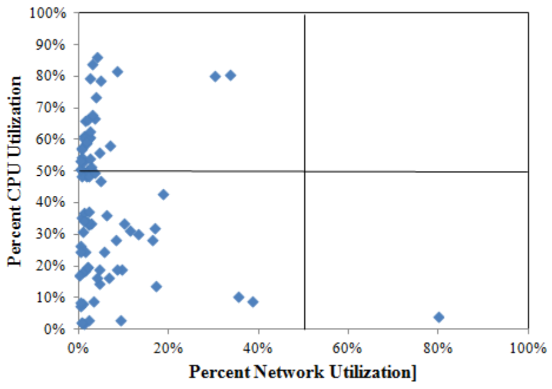
When plotting the resource consumption of two resources R_1 and R_2, we add horizontal and vertical lines, dividing the space into four equally sized quadrants. Queries in the first quadrant, upper right, use both resources R_1 and R_2 intensively; queries located in the second quadrant, upper left, use mostly R_2, queries in the third quadrant, left bottom, use neither R_1 nor R_2 intensively, and queries in the



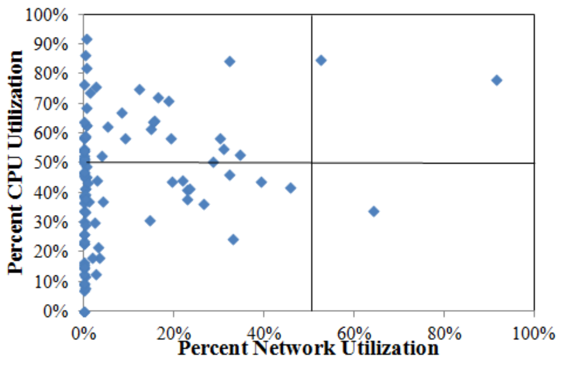
(a) System A



(b) System B



(c) System C



(d) System D

Figure 6.2.18: Network-CPU utilization

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

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 6.2.16a, 6.2.16b, 6.2.16c, and 6.2.16d 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 6.2.17a, 6.2.17b, 6.2.17c, and 6.2.17d 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 seems 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 6.2.18a, 6.2.18b, 6.2.18c, and 6.2.18d 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 that 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.

Chapter 7

Conclusions

In this work, we developed industry standard benchmarks for the decision support domain in three areas: (i) *Internet of Things*, (ii) *Data Integration*, and (iii) *Decision Support*. For the Internet of Things domain, we developed TPCx-IoT, the first industry standard benchmark for measuring the performance of IoT gateway systems. Similar to previous TPC express benchmarks, TPCx-IoT is a kit-based benchmark. It is modeled after a real-world scenario of a power utility provider that must distribute power through a smart grid with many power substations. However, the performance data of TPCx-IoT can be applied to any IoT installation that must ingest data from edge-devices at high speeds, while concurrently performing real-time analytic queries that feed a dash-board for monitoring purposes. We illustrated the power substation use case in detail and showed how the elements of the use case map to elements of the benchmark and parts of the SUT. The TPCx-IoT benchmark was accepted by the TPC in May 2017.

We presented the performance results obtained from three different industry configurations running HBase 1.2.0. The first set of performance experiments scaled the number of power substations from one to 48. It showed that the read and write operations of TPCx-IoT impose a significant workload to show where the limits of a system are. It also showed that the execution rule requirement of executing the workload for at least 1,800s can easily be fulfilled. The second requirement of performing at least 20 $\frac{kvps}{s}$ per sensors showed to be reasonable and is also a good gatekeeper in preventing benchmark sponsors from reducing the read requirements during query execution. The larger the configuration, the more gateways can be supported. We also showed that TPCx-IoT's fixed workload requirement, that is, to ingest a fixed number of *kvps* per power substation, can reveal deficiencies in a system's ability to load-balance data ingestion across multiple power substations.

For the Data Integration domain, we developed TPC-DI, the first industry standard benchmark for data integration. It uses the enterprise benchmark model of the TPC. Like previous enterprise TPC benchmarks, TPC-DI is a technology agnostic, end-to-end benchmark. It is modeled after the real-world scenario of a retail brokerage firm's information system, where five different data sources have to be integrated into a decision support system. The data sources feature different formats, granularities, and constraints in a highly realistic data model. Target systems of the benchmark are DI systems that enable data transformation and integration. The benchmark was accepted by the TPC in January 2014.

For decision support, we developed the second version of TPC-DS, which allows SQL-based big data systems to run and publish TPC-DS benchmark results. 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 of current big data solutions: (i) Our test systems apply various degrees of data pruning, but no system prunes the maximum possible. (ii) Query elapsed times during single-user runs on each system vary between seconds and hours. (iii) Our test systems optimize differently for single- and multi-user runs. Some benefit more, some less. (iv) Our test systems show vastly different resource utilizations for single- and multi-user runs. (v) Many queries show a wide spectrum of resource utilization across the four setups.

Our analysis also revealed some 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). On 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.

DSS continue to evolve which require incremental changes to the existing benchmarks, TPCx-IoT, TPC-DI and TPC-DS. IoT systems are evolving to include smart edge devices. As the compute power of sensors will increase, gateway systems will be able to push data processing into sensors. This will create a two-tier compute architecture that is currently not addressed in TPCx-IoT. Historically, DI processes were run during a maintenance window. Companies require this maintenance window to reduce or be completely eliminated for their DSS to be available 24 by 7. This requires DSS to integrate new data as it arrives or execute queries on partially integrated data. This is a concept not currently addressed in TPC-DI. The analysis of TPC-DS queries done in this work has revealed that the current query mix can be improved by developing new queries and eliminating non-discriminatory queries.

An emerging field in the area of DSS that is yet to be covered by any industry standard benchmarks is deep learning, where machine learning algorithms are deployed to create autonomous systems. With Watson IBM has introduced one of the first commercial products that use deep learning to aid the decision making of companies. As the domain of deep learning matures, and more companies offer commercial products for it, the need for industry standard benchmarks to measure the performance of deep learning solutions will increase. Performing deep learning tasks is data and compute intensive, which aligns well with the mission of the TPC.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] SPEC. *The Software Performance Evaluation Corporation (SPEC).* http://www. businesswire.com/news/home/20070417005047/en /SAS-Smashes-ETL-World-Record-Establishing-New.

[2] S. P. Council. *Storage Performance Council.* http://www.storageperformance.org/home/.

[3] *EMBC.* `http://www.eembc.org/`. EMBC. 2017.

[4] TPC. *Home Page TPC.* `http://www.tpc.org`. 2017.

[5] G. B. Davis and M. H. Olson. *Management Information Systems: Conceptual Foundations, Structure, and Development (2Nd Ed.)* New York, NY, USA: McGraw-Hill, Inc., 1985. ISBN: 0-07-015828-2.

[6] TPC. *TPC EXPRESS BENCHMARK ^{TM} IoT (TPCx-IoT) Standard Specification Version 1.0.0.* `http://www.tpc.org/TPC_Documents_Current_Versions\/pdf\/TPCx\discretionary{-}{}{}IoT_v1.5.x.pdf`. TPCx-IoT specification. 2017.

[7] TPC. *TPC BENCHMARK ^{TM} DI Version 1.1.0.* http://www.tpc.org/TPC_Documents_ Current_ Versions/pdf/TPC-DI_v1.1.0.pdf.

[8] TPC. *TPC BENCHMARK ^{TM} DS Version 2.6.0.* http://www.tpc.org/tpc_documents_ current_ versions/pdf/tpc-ds_v2.6.0.pdf.

[9] O. Serlin. "The History of DebitCredit and the TPC." In: *The Benchmark Handbook.* 1991, pp. 19–38.

[10] C. Ballinger. "TPC-D: Benchmarking for Decision Support." In: *The Benchmark Handbook for Database and Transaction Systems (2nd Edition).* Ed. by J. Gray. Morgan Kaufmann, 1993. ISBN: 1-55860-292-5.

[11] M. Poess, B. Smith, L. Kollár, and P. Larson. "TPC-DS, taking decision support benchmarking to the next level." In: *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6, 2002.* Ed. by M. J. Franklin, B. Moon, and A. Ailamaki. ACM, 2002, pp. 582–587. ISBN: 1-58113-497-5. DOI: `10.1145/564691.564759`. URL: `http://doi.acm.org/10.1145/564691.564759`.

[12] M. Poess and J. M. Stephens. "Generating Thousand Benchmark Queries in Seconds." In: *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*. Ed. by M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer. Morgan Kaufmann, 2004, pp. 1045–1053. ISBN: 0-12-088469-0. URL: `http://www.vldb.org/conf/2004/IND2P3.PDF`.

[13] J. M. Stephens and M. Poess. "MUDD: a multi-dimensional data generator." In: *Proceedings of the Fourth International Workshop on Software and Performance, WOSP 2004, Redwood Shores, California, USA, January 14-16, 2004*. Ed. by J. J. Dujmovic, V. A. F. Almeida, and D. Lea. ACM, 2004, pp. 104–109. DOI: `10.1145/974044.974060`. URL: `http://doi.acm.org/10.1145/974044.974060`.

[14] M. Poess and C. Floyd. "New TPC Benchmarks for Decision Support and Web Commerce." In: *SIGMOD Record* 29.4 (2000), pp. 64–71. DOI: `10.1145/369275.369291`. URL: `http://doi.acm.org/10.1145/369275.369291`.

[15] J. Ericson, G. Rahn, M. Kornaker, and Y. Chen. *Impala Performance Update: Now Reaching DBMS-Class Speed.* `http://blog.cloudera.com/blog/2014/01/impala-performance-qdbms-class-speed/`.

[16] Michael Armbrust, Zongheng Yang. *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`.

[17] S. Harris. *Big SQL 3.0: Hadoop-DS benchmark - Performance isn't everything...* `https://developer.ibm.com/hadoop/blog/2014/12/02/big-sql-3-0-hadoop-ds-benchmark-performance-isnt-everything/`.

[18] P. M. Davis. *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`.

[19] S. 2.0. *Syncsort And Vertica Shatter Database ETL World Record Using HP BladeSystem C-Class.* http://www. itweb.co.za /index.php?option=com _content&view=article&id=89556.

[20] Microsoft. *ETL World Record!* https://blogs.msdn.microsoft.com/sqlperf/2008/02/27/etl-world-record/.

[21] W. S. Computing. *Informatica sets world record in data integration performance.* http://www.itweb.co.za/index.php?option=com_content&view=article&id=118340.

[22] B. Wire. *SAS Smashes ETL World Record While Establishing New, Real-World Benchmarks.* http://www.businesswire.com/news/home/20070417005047/en/SAS-Smashes-ETL-World-Record-Establishing-New.

[23] VoltDB. *Solving the Fast Data Problem for the Cloud Integrating VoltDB and the IBM SoftLayer Cloud Computing Platform Delivers up to 5X the Performance of Amazon Web Services.* https://cdn2.hubspot.net/hubfs/2180197/content/data_sheets/lv_data_sheets/lv-data-sheet-solving-the-fast-data-problem-for-the-cloud-voltdb-softlayer- benchmark.pdf ?t= 1472434607114.

[24] Gartner. *IoT Growth Report By Gartner.* `http://www.gartner.com/newsroom/id/3598917`. SmartMeter. 2017.

[25] J. Gray, ed. *The Benchmark Handbook for Database and Transaction Systems (2nd Edition).* Morgan Kaufmann, 1993. ISBN: 1-55860-292-5.

[26] M. Stonebraker. "A New Direction for TPC?" In: *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers.* Ed. by R. O. Nambiar and M. Poess. Vol. 5895. Lecture Notes in Computer Science. Springer, 2009, pp. 11–17. ISBN: 978-3-642-10423-7. DOI: `10.1007/978-3-642-10424-4_2`. URL: `http://dx.doi.org/10.1007/978-3-642-10424-4_2`.

[27] K. Huppler. "The Art of Building a Good Benchmark." In: *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers.* 2009, pp. 18–30.

[28] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. "Benchmarking cloud serving systems with YCSB." In: *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC 2010, Indianapolis, Indiana, USA, June 10-11, 2010.* Ed. by J. M. Hellerstein, S. Chaudhuri, and M. Rosenblum. ACM, 2010, pp. 143–154. ISBN: 978-1-4503-0036-0. DOI: `10.1145/1807128.1807152`. URL: `http://doi.acm.org/10.1145/1807128.1807152`.

[29] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling.* 2nd. John Wiley and Sons, Inc., 2002.

[30] T. Rabl, M. Frank, H. M. Sergieh, and H. Kosch. "A Data Generator for Cloud-Scale Benchmarking." In: *TPCTC '10.* 2010, pp. 41–56.

[31] A. Crolotte. "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.* 2009, pp. 146–152.

[32] M. Poess, T. Rabl, and H.-A. Jacobsen. "Analysis of TPC-DS: the first standard benchmark for SQL-based big data systems." In: *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24 - 27, 2017.* ACM, 2017, pp. 573–585. ISBN: 978-1-4503-5028-0. DOI: `10.1145/3127479.3128603`. URL: `http://doi.acm.org/10.1145/3127479.3128603`.

[33] M. Poess, T. Rabl, H.-A. Jacobsen, and B. Caufield. "TPC-DI: The First Industry Benchmark for Data Integration." In: *PVLDB* 7.13 (2014), pp. 1367–1378. URL: `http://www.vldb.org/pvldb/vol7/p1367-poess.pdf`.

[34] M. Poess, R. Othayoth, C. Narasimhadevara, K. Kulkarni, T. Rabl, and H.-A. Jacobsen. "Analysis of TPCx-IoT: The First Industry Standard Benchmark for IoT Gateway Systems." In: *Proceedings of the 34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-20, 2018*.

[35] TPC. *About the TPC*. http://www.tpc.org/information/about/abouttpc.asp.

[36] TPC. *BYLAWS OF THE TRANSACTION PROCESSING PERFORMANCE COUNCIL*. http://www.tpc.org/tpc_documents_current_versions/pdf/bylaws_v2.9.0.pdf.

[37] TPC. *TPC Policies*. http://www.tpc.org/tpc_documents_current_versions/pdf/policies_v6.10.0.pdf.

[38] K. Huppler and D. Johnson. "TPC Express - A New Path for TPC Benchmarks." In: *Performance Characterization and Benchmarking - 5th TPC Technology Conference, TPCTC 2013, Trento, Italy, August 26, 2013, Revised Selected Papers*. 2013, pp. 48–60.

[39] K. Huppler. "Price and the TPC." In: *Performance Evaluation, Measurement and Characterization of Complex Systems - Second TPC Technology Conference, TPCTC 2010, Singapore, September 13-17, 2010. Revised Selected Papers*. Ed. by R. O. Nambiar and M. Poess. Vol. 6417. Lecture Notes in Computer Science. Springer, 2010, pp. 73–84. ISBN: 978-3-642-18205-1. DOI: 10.1007/978-3-642-18206-8_6. URL: https://doi.org/10.1007/978-3-642-18206-8%5C_6.

[40] TPC. *TPC Pricing Specification Version 2.1.1*. http://www.tpc.org/TPC_Documents_Current_Versions/pdf/Pricing_v2.1.1.pdf.

[41] J. Gray, ed. *The Benchmark Handbook for Database and Transaction Systems (1st Edition)*. Morgan Kaufmann, 1991.

[42] J. Gray. "Introduction." In: *The Benchmark Handbook*. 1991, pp. 1–17.

[43] *IoTMARK*. http://www.eembc.org/iot-connect/about.php. EMBC. 2017.

[44] M. F. Arlitt, M. Marwah, G. Bellala, A. Shah, J. Healey, and B. Vandiver. "IoTAbench: an Internet of Things Analytics Benchmark." In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, January 31 - February 4, 2015*. Ed. by L. K. John, C. U. Smith, K. Sachs, and C. M. Lladó. ACM, 2015, pp. 133–144. ISBN: 978-1-4503-3248-4. DOI: 10.1145/2668930.2688055. URL: http://doi.acm.org/10.1145/2668930.2688055.

[45] *Smart Meter*. http://en.wikipedia.org/wiki/Smart_meter. SmartMeter. 2017.

[46] Informatica Corporation. *Informatica And Sun Achieve Record-Setting Results In Data Integration Performance And Scalability Test*. http://www.informatica.com/ca/company/news-and-events-calendar/press-releases/06062005d-sun.aspx. 2005.

[47] Syncsort Incorporated. *Syncsort and Vertica Shatter Database ETL World Record Using HP BladeSystem c-Class.* `http://www.prnewswire.co.uk/news-releases/syncsort-and-vertica-shatter-database-etl-world-record-using-hp-bladesystem-c-class-152940915.html`. 2008.

[48] L. Wyatt, T. Shea, and D. Powell. *We Loaded 1TB in 30 Minutes with SSIS, and So Can You.* `http://technet.microsoft.com/en-us/library/dd537533(v=sql.100).aspx`. Microsoft Cooperation. 2009.

[49] Manapps. *ETL Benchmarks.* `https://marcrussel.files.wordpress.com/2008/10/etlbenchmarks_manappsc221008.pdf`. 2008.

[50] P. Vassiliadis, A. Karagiannis, V. Tziovara, and A. Simitsis. "Towards a Benchmark for ETL Workflows." In: *QDB*. 2007, pp. 49–60.

[51] A. Simitsis, P. Vassiliadis, U. Dayal, A. Karagiannis, and V. Tziovara. "Benchmarking ETL Workflows." In: *TPC TC '09*. 2009, pp. 183–198.

[52] R. O. Nambiar and M. Poess. "The Making of TPC-DS." In: *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006.* 2006, pp. 1049–1058.

[53] M. Pöss, R. O. Nambiar, and D. Walrath. "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.* 2007, pp. 1138–1149.

[54] T. Ivanov, T. Rabl, M. Poess, A. Queralt, J. Poelman, N. Poggi, and J. Buell. "Big Data Benchmark Compendium." In: *Proceedings of the Seventh TPC Technology Conference on Performance Evaluation and Benchmarking.* 2015.

[55] T. G. Armstrong, V. Ponnekanti, D. Borthakur, and M. Callaghan. "LinkBench: a Database Benchmark Based on the Facebook Social Graph." In: *SIGMOD*. 2013, pp. 1185–1196.

[56] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang. "Introducing the Graph 500." In: *Cray User's Group (CUG)* (2010).

[57] K. Kim, K. Jeon, H. Han, S.-g. Kim, H. Jung, and H. Yeom. "MRBench: A Benchmark for MapReduce Framework." In: *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on.* Dec. 2008, pp. 11–18.

[58] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura. "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. Ischia, Italy: ACM, 2015, 53:1–53:8. ISBN: 978-1-4503-3358-0.

[59] R. O. Nambiar, M. Poess, A. Dey, P. Cao, T. Magdon-Ismail, D. Q. Ren, and A. Bond. "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.* 2014, pp. 1–12.

[60] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. "BigBench: Towards an Industry Standard Benchmark for Big Data Analytics." In: *SIGMOD*. 2013.

[61] T. Rabl, A. Ghazal, M. Hu, A. Crolotte, F. Raab, M. Poess, and H.-A. Jacobsen. "BigBench Specification V0.1." In: *Specifying Big Data Benchmarks*. Ed. by T. Rabl, M. Poess, C. Baru, and H.-A. Jacobsen. Vol. 8163. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 164–201.

[62] 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*. 2009, pp. 165–178.

[63] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. "The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis." In: *ICDEW*. 2010.

[64] AMP Lab. *AMP Lab Big Data Benchmark*. https://amplab.cs.berkeley.edu/benchmark/. 2013.

[65] M. Adamiak, B. Kasztenny, and W. Premerlani. "Synchrophasors: definition, measurement, and application." In: *Proceedings of the 59th Annual Georgia Tech Protective Relaying, Atlanta, GA* (2005), pp. 27–29.

[66] Bluethooth. *Core Version 5.0*. `https://www.bluetooth.com/specifications/bluetooth-core-specification`. Bluetooth Standard. 2017.

[67] IEEE. *802.15.4-2011 - IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. `http://standards.ieee.org/findstds/standard/802.15.4-2011.html`. 2017.

[68] IEEE. *IEEE 802.11TM WIRELESS LOCAL AREA NETWORKS*. `http://www.ieee802.org/11/`. 2017.

[69] E. T. S. I. (ETSI). *Mobile technologies GSM*. `http://www.etsi.org/technologies-clusters/technologies/mobile/gsm`. 2017.

[70] OASIS. *MQTT Version 3.1.1 Plus Errata 01*. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html`. MQTT Standard. 2017.

[71] TPC. *TPCx-IoT first standard for IOT Gateway systems*. `http://www.tpc.org/tpcx-iot/default.asp`. 2017.

[72] C. E. Commission. *Energy Infrastructure: Statewide Operational Substations*. `http://www.energy.ca.gov/maps/powerplants/Operational_Substations.xlsx`. 2017.

[73] P. Gas and Electric. *Pacific Gas and Electric Company Company Profile*. `https://www.pge.com/en_US/about-pge/company-information/profile/profile.page`. 2017.

[74] S. Selvam. "Efficient Monitoring in Fossil-Fueled Power Plants using Low-Cost Wireless Sensors." In: 2015, pp. 2319–3344.

[75] tefDesign. *PG&E Larkin Substation*. `http://tefarch.com/projects/detail/64/`. 2017.

[76] S. F. P. Department. *320-400 Paul Avenue Data Center and associated Extension of PG&E 12kV Electrical Distribution Circuits.* `http://sfmea.sfplanning.org/2011.0408E_FMND.pdf`. 2014.

[77] E. P. R. Institute. *Sensor Technologies for a Smart Transmission System.* `http://www.remotemagazine.com/images/EPRI-WP.pdf`. 2009.

[78] G. F. Fine, L. M. Cavanagh, A. Afonja, and R. Binions. "Metal Oxide Semi-Conductor Gas Sensors in Environmental Monitoring." In: *Sensors* 10.6 (2010), pp. 5469–5502. ISSN: 1424-8220. DOI: `10.3390/s100605469`. URL: `http://www.mdpi.com/1424-8220/10/6/5469`.

[79] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. "Bigtable: A Distributed Storage System for Structured Data (Awarded Best Paper!)" In: *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA.* Ed. by B. N. Bershad and J. C. Mogul. USENIX Association, 2006, pp. 205–218. ISBN: 1-931971-47-1. URL: `http://www.usenix.org/events/osdi06/tech/chang.html`.

[80] Apache. *Apache HBase homepage.* `https://hbase.apache.org/`. 2017.

[81] Google. *Overview of Internet of Things.* https://cloud.google.com/solutions/iot-overview.

[82] Microsoft. *Overview of Azure IoT Suite.* https://docs.microsoft.com/en-us/azure/iot-suite/iot-suite-overview.

[83] SAS Institute Inc. *New Release of SAS Enterprise ETL Server Sets Performance World Record.* `http://callcenterinfo.tmcnet.com/news/2005/mar/1126716.htm`. 2005.

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

[85] T. Rabl and M. Poess. "Parallel data generation for performance analysis of large, complex RDBMS." In: *DBTest '11.* 2011, p. 5.

[86] M. Frank, M. Poess, and T. Rabl. "Efficient Update Data Generation for DBMS Benchmark." In: *ICPE '12.* 2012.

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

[88] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[89] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. "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.* 2008, pp. 1099–1110.

[90] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. “Hive: A Warehousing Solution Over a Map-Reduce Framework.” In: *PVLDB* 2.2 (2009), pp. 1626–1629.

[91] Hortonworks. *Stinger.* `http://hortonworks.com/innovation/stinger/`.

[92] Cloudera. *Impala.* `http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html`.

[93] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. “Shark: SQL and Rich Analytics at Scale.” In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of data.* ACM. 2013, pp. 13–24.

[94] Facebook. *Presto.* `https://prestodb.io/`.

[95] *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`. IBM. 2017.

[96] *Oracle Big Data SQL.* `https://www.oracle.com/database/big-data-sql/index.html`. Oracle Corporation. 2017.

[97] *SAP Vora.* `https://www.sap.com/products/hana-vora-hadoop.html`. SAP. 2017.

[98] Transaction Processing Performance Council. *Specification TPC-DS Version 2.1.* `http://www.tpc.org/tpcds/default.asp`. 2015.

[99] Transaction Processing Performance Council. “Top 10 TPC-H publications grouped by scale factor.” In: (July 2016). URL: `http://www.tpc.org/tpch/results/tpch_perf_results.asp`.

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

[101] R. Tibshirani, G. Walther, and T. Hastie. “Estimating the number of clusters in a data set via the gap statistic.” In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2 (2001), pp. 411–423. ISSN: 1467-9868. DOI: `10.1111/1467-9868.00293`. URL: `http://dx.doi.org/10.1111/1467-9868.00293`.