TPC-DI: The First Industry Benchmark for Data Integration

Meikel Poess
Server Technologies
Oracle Corporation
Redwood Shores,California
USA
mpoess@oracle.com

Tilmann Rabl,Hans-Arno Jacobsen
Middleware Systems
Research Group
University of Toronto
Canada
tilmann.rabl@utoronto.ca,
jacobsen@eecg.toronto.edu

Brian Caufield
InfoSphere DataStage
IBM
USA
bcaufiel@us.ibm.com

ABSTRACT

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 the 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 store. This is done in the context of a variety of scenarios, such as 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. With these scenarios relying on up-to-date information it is critical to implement a highly 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 having a highly performing DI system, there has been no industry standard for measuring and comparing their performance. The TPC, acknowledging this void, has released TPC-DI, an innovative benchmark for data integration. This paper motivates the reasons behind its development, describes its main characteristics including workload, run rules, metric, and explains key decisions.

1. INTRODUCTION

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, todays 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 practice in which organizations make performance claims based on self-designed, highly biased benchmarks. Today, a large number of *world record* claims have been made for DI systems (e.g., [8, 10, 12]). 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:

[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`

- 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.,
- 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.

Following TPC's core philosophy, TPC-DI is technology agnostic, i.e., TPC-DI defines a set of functional requirements that can be run on any DI system, regardless of specific hardware or software. This enables the performance evaluation of a broad spectrum of hardware and software tools to be conducted in a fair and open way. However, it also increases the complexity of developing a benchmark specification as all functional requirements, such as specific transformations, 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 its own implementation of the TPC-DI specification and to submit proof that the implementation meets all benchmark requirements.

The contributions of this paper can be summarized as follows: TPC-DI is the first industry standard benchmark that enables hardware and software vendors, hereafter referred to as benchmark sponsors, to showcase the performance, price-performance, and energy efficiency of their systems in a comprehensive, competitive, fair, and open way. The benchmark specification as well as a detailed disclosure report for every benchmark is available for anybody to verify the results. With benchmark TPC-DI results becoming available customers, who would like to evaluate different DI solutions under a controlled workload, are able to do so. This paper describes the entire benchmark in a deep and comprehensive way, discusses alternative designs where possible and gives reasons behind design decisions. It further analyzes the workload and presents results obtained on a test system.

The remainder of this paper is structured as follows. Section 2 introduces the source and target data model. Section 3 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 4, the transformations of the DI workload are explained. They form the core workload of the benchmark. Section 5 presents the execution rules and metric used to govern how the transformations have to be executed, timed, and weighted to compute the ranking of DI systems. Special emphasis is put on explaining the metric and the decisions that led to its definition. A performance study is presented in Section 6. The paper concludes with future work in Section 8.

2. 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, i.e., 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 [3]. 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 1 outlines the conceptual model of the TPC-DI benchmark. The top portion displays the five 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 by the *DeWitt Clause* 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 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, e.g., 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 4. 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*.

2.1 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, i.e., *incremental updates*. The concepts of historical load and incremental updates are described comprehensively in Section 5.2. 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.

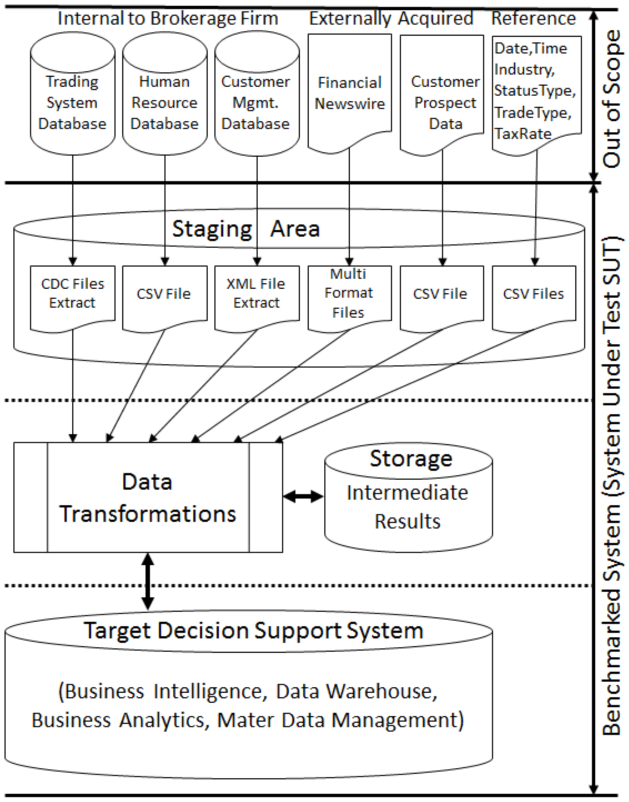
Internal to Brokerage Firm
Externally Acquired
Reference
Trading System Database
Human Resource Database
Customer Mgmt. Database
Financial Newswire
Customer Prospect Data
Date,Time Industry, StatusType, TradeType, TaxRate
Out of Scope
Staging Area
CDC Files Extract
CSV File
XML File Extract
Multi Format Files
CSV File
CSV Files
Data Transformations
Storage Intermediate Results
Target Decision Support System
(Business Intelligence, Data Warehouse, Business Analytics, Mater Data Management)
Benchmarked System (System Under Test SUT)

Figure 1: Benchmarked System and Workflow

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.

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 (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, i.e. 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.

[6]day is the refresh interval for incremental updates

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, i.e. *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.

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, i.e., 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, e.g. 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 [6]. More on the data generator in Section 3.3.

Table 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.

2.2 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 [5]. 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

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 1: Source Files with Type (DEL=Full Data Dump, CDC=Change Data Capture, XML=Extensible Markup Language, CSV=Comma Separated Value) and Load Usage

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 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.

3. 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 scal-

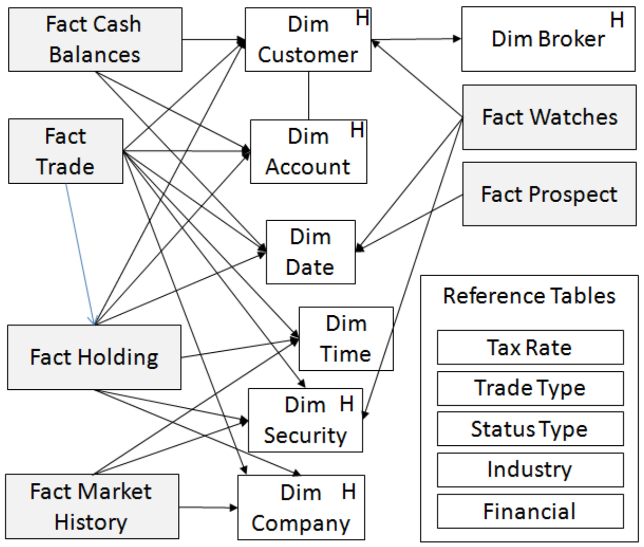

Figure 2: Target Data Schema

ing. 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 TPCs 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.

3.1 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, i.e., 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 [7]. 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, e.g., 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

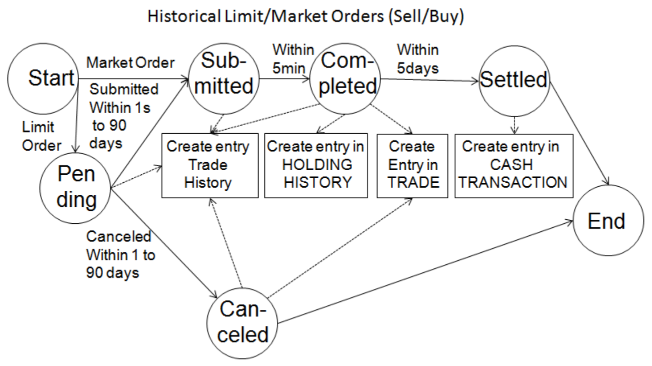
Historical Limit/Market Orders (Sell/Buy)
Start
Market Order
Submitted Within 1s to 90 days
Limit Order
Sub-mitted
Within 5min
Com-pleted
Within 5days
Settled
Pen ding
Create entry Trade History
Create entry in HOLDING HISTORY
Create Entry in TRADE
Create entry in CASH TRANSACTION
End
Canceled Within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

Figure 3: State Diagram for the Order Data Creation of the Historical Load

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, e.g., security.txt and customer.txt need to be consulted to assure that only valid customers trade existing securities, which is not trivial as 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 3 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.

3.2 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, e.g., number of customers or number of ticker symbols. There are two approaches in defining SF:(i) continuous scaling, i.e. $SF \in \mathbb{N}$, or (ii) fixed scaling, i.e., 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, i.e., 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.

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 2: Reference Source Files Size and Rowcount Information

TPC-DI uses continuous scaling based on the number of customers of the fictitious 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 tables, such as date and time, do not scale with

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 3: Source Files Scaling Information

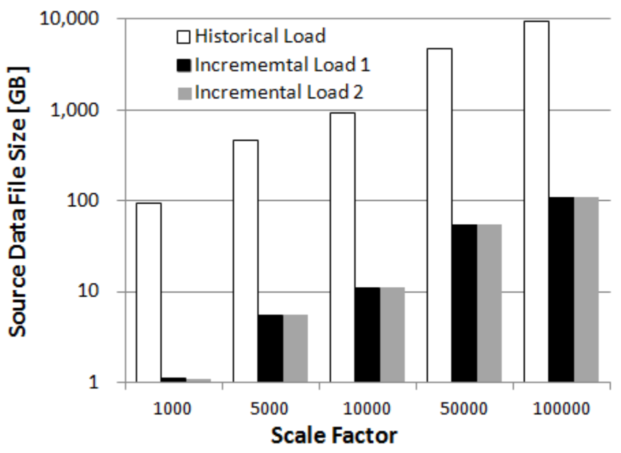

Figure 4: Aggregated Source Data Sizes for Historical and Incremental Loads in [GB]

SF, they remain static.

Table 3.2 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 3.2 summarizes the sizes of all static tables.

3.3 Data Generation with PDGF

Since its first incarnation, the parallel data generation framework PDGF, which was developed at the University of Passau [6], has been improved and extended with many features. It's 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. 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, i.e. 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 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 [1]. 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 [1].

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 3. 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.

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 (e.g., 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

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

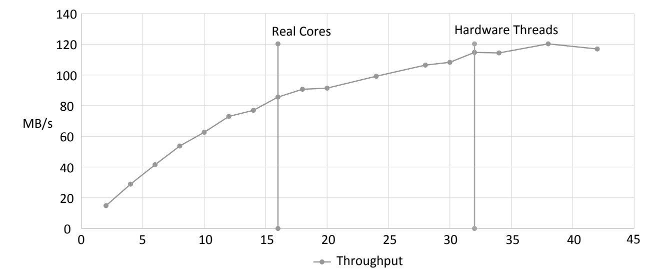

Figure 5: DiGen Scale-Out Performance

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 shows the scale-out performance of DiGen. We generated data for scale factor 100 on a system with 2 E5-2450 Intel CPUs,i.e. 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.

4. 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 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 4. 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, i.e., get the same result and do not over-optimize or cut corners, TPC-DI defines a qualification test. It provides an input data set, i.e., 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 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.

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 4: Transformation characteristics

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, e.g., 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 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.

4.1 History Keeping Dimensions

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: Transformations and their characteristics

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, i.e. 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:

- Update the existing dimension table record for that natural key where IsCurrent is set to TRUE (these updates are known as Expiring the record): (i) The current indicator field, IsCurrent, is set to FALSE to indicate that this is no longer the current record corresponding to the natural key and (ii) The EndDate field is set to the EffectiveDate of the new record.
- 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.

4.2 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, i.e., 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 differenciate between the six different actions associated with customers and accounts, i.e. *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

[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;
else
    Error Out ← IsCurrent, EffectiveDate, EndDate according
    to EffectiveDate being assigned the date value from the
    date portion of @ActionDate;
    BatchID=CurrentBacth
end
```
Algorithm 1: DimAccount Historical Load Transformation

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 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, i.e., 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. METRIC AND EXECUTION RULES

The 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. We describe metrics and execution rules in one section since they are intrinsically connected to each other and they are equally powerful in how they control 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 remains 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 when running the benchmark. The following Section 5.1 describes TPC-DI's execution rules followed by Section 5.2, which describes the TPC-DI's metrics.

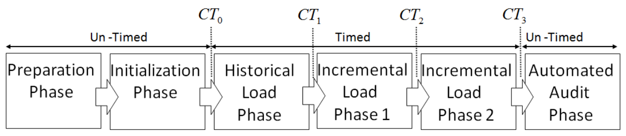

Figure 6: Execution Phases and Metric

5.1 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, e.g. 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 un-timed 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 (v) Automated Audit Phase - untimed

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.

5.2 Metric

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 is 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 (e.g. A, B, C), system comparisons are difficult because vendors can claim they perform well on some of the metrics (e.g. A and C). This might still be acceptable if all components are equally important, but without this determination, there would be much debate on this issue. In order to unambiguously rank results, the TPC benchmarks focus on a single primary performance metric, which encompass all aspects of a systems 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 (e.g. M=f(A,B,C)). Consequently, the TPCs performance metrics measure 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.

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 6 TPC-DI defines four completion time stamps (CTs) to be taken at a precision of 0.1 second (rounded up), e.g. 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 $i \in \{1, 2\}$.

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

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

$$TPC\_DI\_RPS = \lfloor(\sqrt{T_H, min(T_{I1}, T_{I2}}\rfloor \quad (3)$$

5.3 Metric Discussion

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, i.e. 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 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 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

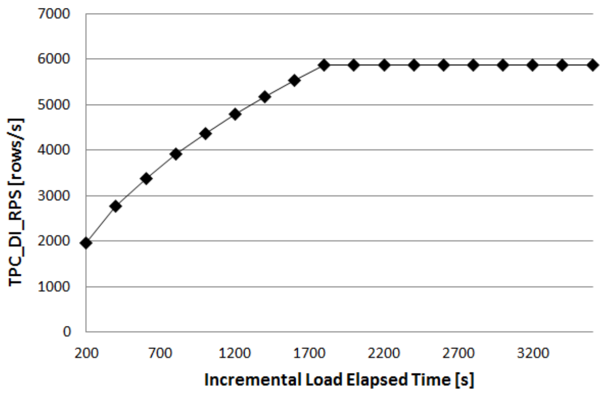

Figure 7: Scaling with Incremental Load Time

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, i.e. $T_{Ii}(1) = T_{Ii}$ 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 7 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, i.e. 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 no negative performance effect of executing multiple incremental load phases, i.e. 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 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), i.e. a system with double the number of resources, e.g. 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 8 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

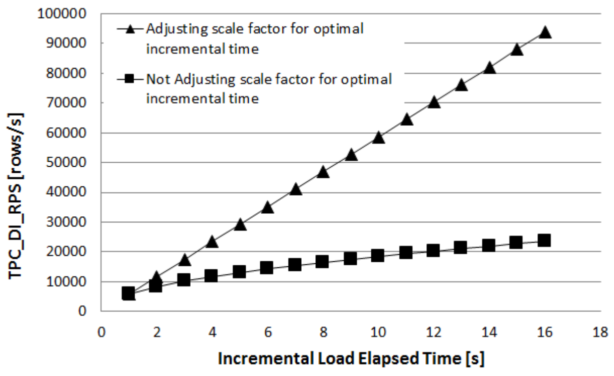

Figure 8: Linear Scaling with Adjusted Elapsed Time

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.

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.

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 N 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 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, i.e. it allows for scalable implementations to be created.

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

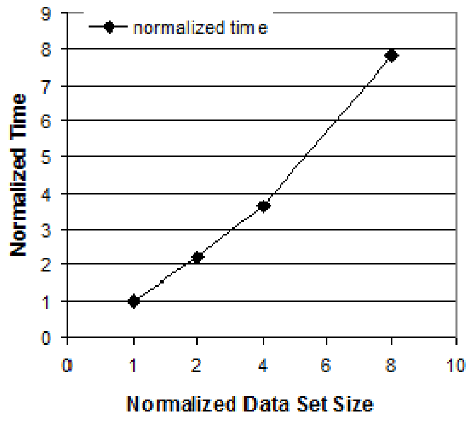

Figure 9: Total Elapsed Time Scaling

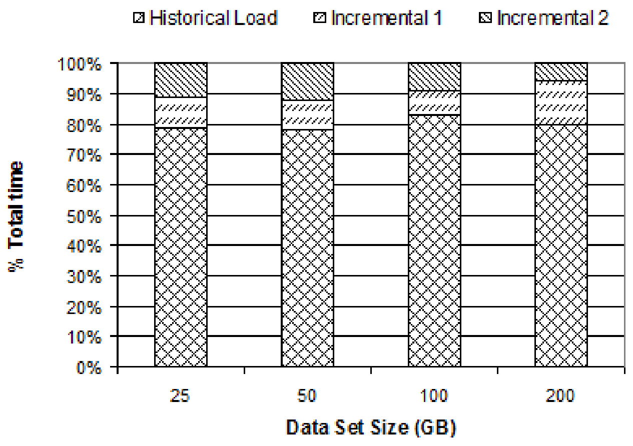

Figure 10: Relative Phase Time

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 N 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 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.

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

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 6: TBD

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, i.e. 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 sponsors 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.

7. RELATED WORK

There has been little research for benchmarks for ETL systems. Today, most systems are tested with rather simple workloads such as loading TPC-H data (c.f., [2, 10, 12]). A more involved proposal was presented by Manapps [4]. In this benchmark, 11 independent ETL jobs were used to compare 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 to understand the bottlenecks in a ETL workload, it does not necessary related 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 [11]. These were extended to a complete ETL workload based on TPC-H [9]. 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.

8. CONCLUSION

In this paper, we present TPC-DI, the first industry standard benchmark for data integration. Like previous TPC benchmarks, TPC-DI is a technology agnostic, end-to-end benchmark 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.

9. REFERENCES

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

[2] 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.

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

[4] Manapps. Etl benchmarks. Technical report, 2008.

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

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

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

[8] SAS Institute Inc. New release of sas enterprise etl server sets performance world record. `http://callcenterinfo.tmcnet.com/news/2005/mar/1126716.htm`, 2005.

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

[10] 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.

[11] P. Vassiliadis, A. Karagiannis, V. Tziovara, and A. Simitsis. Towards a benchmark for etl workflows. In *QDB*, pages 49–60, 2007.

[12] 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`, 2009. Microsoft Cooperation.