Implementing TPC-E, A Streaming Benchmark

By

Brigitte Harder

May 2014

Table of Contents

Introduction

A main memory database (MMDB) is a database where the data is stored in main memory. This increases data access speeds by removing the rate-limiting step of accessing data from a disk. MMDB systems drastically improve performance by employing efficient algorithms that execute fewer instructions[1].

The rise of the multi-core CPUs has made MMDB systems increasingly feasible since overall performance can be improved by dividing an application across cores, although the relationship is not linear[2,3].

The four properties used to guarantee reliable database processing are atomicity, consistency, isolation, and durability (ACID)[4]. MMDB systems support atomicity, consistency, and isolation; however, they do not support durability. This is because MMDB systems are volatile as the information in the database is stored in the main memory of a computer (RAM). Thus, if the computer is turned off, RAM will be erased and all information lost. New systems are being developed to create a type of RAM that is non-volatile and will not be wiped when a computer is shut down. With the development of these new systems, the MMDB will at some point become durable, and therefore, will be deemed a reliable processing system.

H-Store

H-Store is a MMDB system that runs on shared-nothing nodes in order to optimize execution of on-line transaction processing (OLTP) applications. It is both parallel and distributed[5]. The structure of H-Store is depicted in Figure 1.[6]

As Figure 1 illustrates, the typical H-Store node, computer, has multiple CPUs; each site is assigned to execute on one processor. Thus, one node can have multiple sites, and each site is independent of the others, in other words, no disk space or memory is shared. H-Store makes use of this multi-core structure to get faster and more efficient transaction processing throughput than traditional database systems.

H-Store is written in C++ and Java. The Execution Engines (EEs) of H-Store are in charge of the low-level data manipulation operations in the database. Every query

executed in H-Store invokes the code in the Execution Engine thread. The EE is written in C++, and the front end that contains the core functional features is written in Java. Java Native Interface (JNI) is used to call the C++ code functions of the EE from Java code level[7].

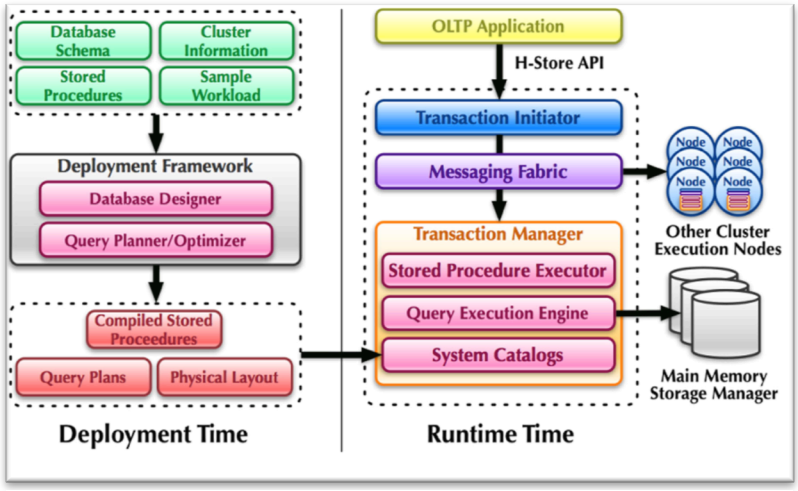

Figure 1: H-Store system architecture

TPC-E

TPC Benchmark™ E (TPC-E) was developed by the Transaction Processing Performance Council (TPC). TPC produces benchmarks that measure transaction processing (TP) and database (DB) performance in terms of how many transactions a given system and database can perform per unit of time, e.g., transactions per second or transactions per minute[7].

As illustrated in Figure 2, the TPC-E benchmark simulates the OLTP workload of a brokerage firm. A central database executes transactions related to the firm's customer accounts. Different transaction types are defined to simulate the interactions of the firm with its customers as well as its business partners. Different transaction types have varying run-time requirements.

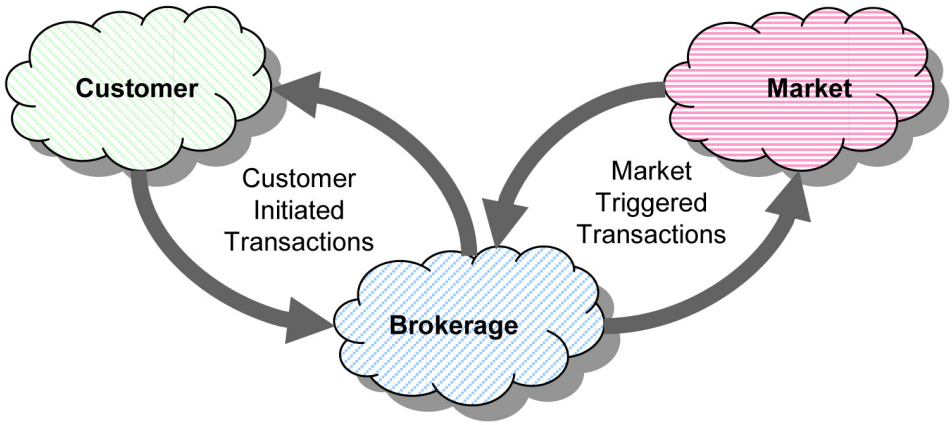

Figure 2: TPC-E model

TPC-E Business Model

The TPC-E benchmark uses a database to model a brokerage firm. Customers generate transactions, including trades, account inquiries, and market research. The brokerage firm transacts trades on behalf of customers in designated financial markets and executes orders and updates accounts. The benchmark is composed of a set of transactions that are executed against three sets of database tables representing market data, customer data, and broker data. Another set of tables contains generic descriptive data such as addresses and zip codes.

TPC-E models the activity of brokerage firm that must manage customer accounts, execute customer Trade-Orders, and be responsible for the interactions of customers with financial markets. Figure 3 illustrates the transaction flow[8].

TPCE results are given in transactions per second (tps) and refer to the number of Trade-Result transactions the server can sustain over a period of time. TPC-E is considered to be broadly representative of modern OLTP systems.

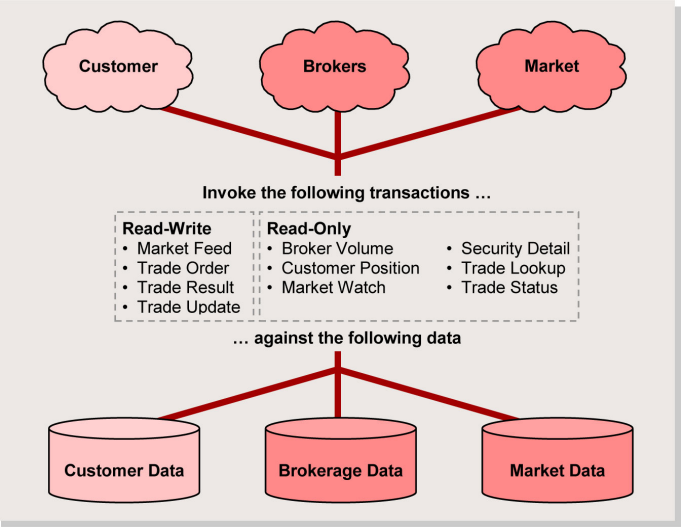

Figure 3: TPC-E Structure

Project Objective

The objective of this project was to continue earlier work that had been done to implement TPC-E and move towards the final goal of implementing TPC-E. It is believed that TPC-E would be a useful benchmark to evaluate the performance of S-Store, a streaming system built on top of H-Store. As S-Store is new and serves a different purpose than H-Store, new benchmarks are being developed to measure how quickly S-Store functions, and how well it works as a streaming system. TPC-E is an appropriate benchmark for S-Store as it is based on a brokerage firm with streaming transactions.

Implementing TPC-E has been an ongoing project, and various people have contributed to implementing it over time. As of November, when I began discussing the possibility of working on TPC-E, the implementation had been abandoned before being completed due to too much complexity within the benchmark and too many errors within the code.

Initially, I stated my preference to restart the implementation entirely from the beginning since the existing software was only partially completed and not having been tested, it had many unresolved bugs. Since no one knew or understood how the software worked, there was no information about whether the problems would even be resolvable. As the work had previously been abandoned due to too many errors, it seemed to me that rewriting the implementation entirely was preferable and would at least ensure a clean

implementation. Nevertheless, the decision was made to use the software anyway with the understanding that it might fail and a rewrite might be required at a later point in time. Therefore, using the partially completed work meant that the project involved debugging, testing, and overlaying new code on existing non-working files.

After reviewing the work that had been done, I decided to proceed by scaling the program down to the minimum necessary to perform as a useful benchmark. At that time, I was told to that TPC-E for H-Store would be implemented first and then modified later to be compatible with S-Store.

Prior to starting any coding, I researched and obtained approval to make the initial adjustments to the benchmark program. The idea was to simplify the process as much as possible, and then add back in the less essential constructs once the implementation was complete and values were being returned. I reduced the transactions to the essential four: Trade-Order, Market-Watch, Market-Feed, and Trade-Result. I believed these four procedures would sufficiently demonstrate the quality of the streaming system while still modeling the basic components of the brokerage firm.

I then selected the tables for implementation. There were 33 tables in the original benchmark and the ER diagram was extremely convoluted. Tables were reduced to an essential 17: Status_Type, Company, Broker, Customer_Info, Trade, Trade_Request, Trade_History, Trade_Type, Holding, Holding_History, Holding_Summary, Cash_Transaction, Last_Trade, and Daily_Market.

Within these tables, further changes were made:

1) Foreign keys were removed;

2) Customer and Customer_Accounts were merged into Customer_Info;

3) Authorization of other users was removed, meaning only an authorized customer had access to their account;

4) Tax calculations were removed and given a constant factor;

5) Broker's commission calculations were removed and commission was held constant;

6) Trades were paid in cash so settlement was immediate.

Strategies and Challenges

The implementation was set up so that each part would be fixed, tested, and once it was functioning as expected, it would be considered implemented. The work was divided into six parts: 1) TPC-E Loader; 2) Drivers; 3) Customer Emulator (CE); 4) Trade-Order; 5) Market-Watch; 6) Market Exchange Emulator (MEE); 7) Market-Feed; and 8) Trade-Result. None of these systems were functioning when I was given the files.

TPC-E Loader

H-Store files had to be modified in order to run TPC-E. This was done by creating and implementing a project builder so that the benchmark would actually execute. Next, all the data tables had to be correctly populated and required modifying the SQL of the tables, as well as modifying the data generators that populated the tables. Some data generators needed to be removed, and others such as the customer information generator needed to be modified slightly produce the correct account number. I mention this generator in particular because at first the account ID's it generated did not match the account IDs being inserted into other tables such as Trade. I fixed this; TPC-E Loader now functions as expected.

Trade-Generator, as well as other generators, had to be modified to refuse settlement. This involved quite a bit of debug work as the process followed when a trade was paid had to be changed. Other minor changes were made to the generators to incorporate the simplifications, and then testing was required to ensure data integrity.

All the generators now functions as expected.

Drivers

The Client Driver required changes, and although this particular part was not an arduous task, it was an important one nonetheless since the drivers are where the CE and MEE objects are created, correct input is set up, and transactions are executed. These had to be modified so that references to transactions not in use were stripped. Then, later in the project, the TPC-E Client was modified to take in the table information returned for Trade-Order and send it back to the MEE. All drivers have been modified and are now functioning as expected.

Customer Emulator (CE)

The purpose of the CE is to model customers and their requests to the brokerage house. This involves initiating model transactions that would normally be initiated by customers or brokers. These are Trade-Order and Market-Watch. Thus the job of the CE is to create a balanced mix of these transactions, generate the input for these transactions, send the transaction request to the server-under-test (SUT), receive the response from the transaction, and track the execution and response times. The CE has many component files and all had to be changed to remove methods and variables for the CE transactions no longer in use. However, this also involved correctly modifying the equal mixing of transactions, and modifying the transaction input to fit with the new tables and data being used – as will be discussed below.

Trade-Order

The Trade-Order Transaction is designed to emulate the process of ordering a trade. As the benchmark is meant to model a brokerage firm, a customer or broker can place an order. (Note that since the ability to store authorized users was removed, commands that involved ensuring the customer had the right to make a trade were removed as well.) The transaction allows for buys or sells at the current stock price or limit buys and sells. The transaction also provides an estimate of the financial impact of the proposed trade by providing profit/loss data. This allowed the trader to evaluate the desirability of the proposed security trade before either submitting or canceling the trade.

Inputs to the Trade-Order transactions are generated by the CE. There were three major changes made. First, the second frame which ensured the executor had the correct permissions was removed. Second, all stocks to be traded were passed in by symbol only. Since the original transaction was developed to accept an issue number and a company id, it was changed to accept a stock symbol as input. Third, the process of returning the VoltTable Trade-Order was made to return all the information about the trade so that the request could be submitted. This will be discussed later with the MEE.

Trade-Order has been completed and functions as expected.

Market-Watch

The Market-Watch Transaction monitors the overall performance of the market by allowing a customer to track the current daily trend (up or down) of a collection of securities. The collection of securities being monitored may be based upon a customer's current holdings, a customer's watch list of prospective securities, or a particular industry. There were almost no modifications made except to give customers data in real time rather than in advance. Market-Watch was debugged and tested and functions as expected.

Market Exchange Emulator (MEE)

In contrast to the CE, the MEE is responsible for emulating the execution of trades and tracking market activity. In TPC-E the CE sends requests to SUT, which returns results. Those results are then passed to the MEE which returns the final results. So the MEE receives trade requests and corresponding data from the SUT, and then uses this data to initiate the Market-Feed and Trade-Result transactions. It also measures the execution and response time for these transactions and sent the associated data to the SUT.

The MEE of the original TPC-E implementation was incomplete had not been tested; it took months to debug. The problems arose because EGen provides only C++ files for the MEE and CE. Translation from C++ to Java is not linear, and all MEE code as translated had to be compared to the original C++ code and corrected. For example, there were small bugs such as an "=" where it should be "<" and large ones such as errors with pointers.

Once the MEE code was completely re-translated and all errors corrected, it still did not work. After some investigation, I discovered that no data was being sent to the MEE, and since EGen does not show or describe how to the program sends data to the MEE, this was a lengthy process. EGen only creates a SendToMarket object at the end of Trade-Order. First of all, all H-Store transactions must return an approved VoltTable and cannot return Objects of another sort. In addition, passing in a SendToMarket object did not work since H-Store transactions can only have approved VoltTypes as well as primitives passed in as input. Therefore, connecting the data output from the CE to the MEE proved more difficult than I expected.

To resolve the problem, I decided to modify the table being returned from Trade-Order so that it contained the data needed to create a TradeRequest. This data could then be accessed in the TPC-E Client and stored in a Trade-Request object, which could then call submitTradeRequest() to initiate the MEE. This also proved more difficult than expected. First, the function required to execute the transaction so that completed Trade-Order transactions no longer returned a status. This was to prevent the appearance that a transaction was not executed when in fact, it had. Second, accessing the data from the table proved difficult.

However, in the end I was able to modify Trade-Order and TPC-E Client so that a trade Request was submitted to the MEE after each Trade-Order and in this way, input was generated for Trade-Result and Market-Feed. To make the TPC-E Client useable for all the transactions, since data returned from Trade-Result and Market-Feed was no longer needed, transactions were assigned a number on execution. If it was one, it was a Trade-Order and executed and set up a Trade-Request. Otherwise, it was an MEE transaction and executed as designed.

The MEE is complete and now functions as expected.

Market-Feed

The Market-Feed transaction is designed to track the current market information. It updates last trade and will process and limit buys or sells that are waiting for a current price. Market-Feed debugging involved some SQL and input fixes so that types matched. Market-Feed functions as expected.

Trade-Result

The final transaction to be implemented was Trade-Result, the final part of completing a trade. The purpose of Trade-Result is to replace the previously estimated pricing information from Trade-Order with the actual information about the cost/profit of the trade. Once completed, the holdings are updated to reflect the proper information.

Simplifications included removing accessing extra customer information, tax information and commission information. The software also had to be fixed since the original inputs were of a different type than what was required.

Trade-Result is not complete. Trades are being inserted into Holding, Holding_History, and Holding Summary multiple times resulting in errors being thrown as the primary key constraint is violated. I spent many hours attempting to debug this by tracing through the benchmark many times but found no solution. I do not believe that changing the primary key is an option and I exhausted several work around solutions before running out of time.

To resolve this error and complete the implementation, I suggest allowing the student who continues to contact the programmer who started the implementation or TPC when problems arise. I believe that the implementation could have been completed months ago if I had been offered a resource who understood TPC-E and how it functions. All of the problems I encountered could have been resolved much more easily if I was allowed to contact people who knew and understood TPC-E.

Conclusion

The purpose of the project was to continue implementing TPC-E, and the implementation is almost complete. Reflecting on the project, I believe that the person carrying on after me should have some essential information and resources. First, it would have been helpful to know that access on the MIT mainframe was available as the program is too big and slow to run on a personal computer. Second, it would have been more efficient to implement TCP-E with new code. Using incomplete software made the implementation considerably more difficult since errors in the translation were compounded by errors in logic. I believe starting over should be considered by the person carrying on since the final bug I encountered requires changes to the primary key, which will impact the rest of the implementation. Third, the student should be allowed to contact all available resources for assistance as they are likely to understand the implications of changes on the system. For example, several times I was required to go back and change code and try a new fix because I had not realized that a fixing one thing would mean another part of the program would not execute.

Sources

[1] A. Silberschatz, H. F. Korth,and S. Sudarshan, *Database System Concepts.* 6th ed. McGraw-Hill, 2010

[2] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden, "Speedy Transactions in Multicore In-Memory Databases," *ACM SOSP,* Nov. 2013.

[3] L. Qiao, V. Raman, F. Reiss, P. Haas, G. Lohman, "Main Memory Scan Sharing For MultiCore CPUs," *PVLDB,* Aug. 2008.

[4] T. Haerder, and A. Reuter, "Principles of transaction-oriented database recovery". *ACM Computing Surveys,* vol. 15, No. 4, pp, 287-317, Dec. 1983.

[5] Brown University, CMU, MIT, and Yale, "H-Store." http://H-Store.cs.brown.edu/, May 4, 2014 [May 3, 2014].

[6] S. Zdonik, et al., "H-Store: A High Performance, Distributed Main Memory Transaction Processing System" *VLDB,* Aug 2008.

[7] Transaction Processing Performance Council, "TPC." http://www.tpc.org, 2014 [May3, 2014].

[8]TPCE., "TPC BENCHMARK ™ E Standard Specification Version 1.13.0," Apr. 2014.