H-Store: A High-Performance, Distributed Main Memory Transaction Processing System

Robert Kallman
Hideaki Kimura
Jonathan Natkins
Andrew Pavlo
Alexander Rasin
Stanley Zdonik
Brown University
{rkallman, hkimura, jnatkins, pavlo, alexr, sbz}@cs.brown.edu

Evan P. C. Jones
Samuel Madden
Michael Stonebraker
Yang Zhang
Massachusetts Institute of Technology
{evanj, madden, stonebraker, yang}@csail.mit.edu

John Hugg
Vertica Inc.
jhugg@vertica.com

Daniel J. Abadi
Yale University
dna@cs.yale.edu

ABSTRACT

Our previous work has shown that architectural and application shifts have resulted in modern OLTP databases increasingly falling short of optimal performance [10]. In particular, the availability of multiple-cores, the abundance of main memory, the lack of user stalls, and the dominant use of stored procedures are factors that portend a clean-slate redesign of RDBMSs. This previous work showed that such a redesign has the potential to outperform legacy OLTP databases by a significant factor. These results, however, were obtained using a bare-bones prototype that was developed just to demonstrate the potential of such a system. We have since set out to design a more complete execution platform, and to implement some of the ideas presented in the original paper. Our demonstration presented here provides insight on the development of a distributed main memory OLTP database and allows for the further study of the challenges inherent in this operating environment.

1. INTRODUCTION

The use of specialized data engines has been shown to outperform traditional or "one size fits all" database systems [8, 9]. Many of these traditional systems use a myriad of architectural components inherited from the original System R database, regardless if the target application domain actually needs such unwieldy techniques [1]. For example, the workloads for on-line transaction processing (OLTP) systems have particular properties, such as repetitive and short-lived transaction executions, that are hindered by the I/O performance of legacy RDBMS platforms. One obvious strategy to mitigate this problem is to scale a system horizontally by partitioning both the data and processing responsibilities across multiple shared-nothing machines. Although some RDBMS platforms now provide support for this paradigm in their execution framework, research shows that building a new OLTP system that is optimized from its inception for a distributed environment is advantageous over retrofitting an existing RDBMS [2].

Using a disk-oriented RDBMS is another key bottleneck in OLTP databases. All but the very largest OLTP applications are able to fit their entire data set into the memory of a modern shared-nothing cluster of server machines. Therefore, disk-oriented storage and indexing structures are unnecessary when the entire database is able to reside strictly in memory. There are some main memory database systems available today, but again many of these systems inherit the architectural baggage of System R [6]. Other distributed main memory databases have also focused on the migration of legacy architectural features to this environment [4].

Our research is focused on developing *H-Store*, a next-generation OLTP system that operates on a distributed cluster of shared-nothing machines where the data resides entirely in main memory. The system model is based on the coordination of multiple single-threaded engines to provide more efficient execution of OLTP transactions. An earlier prototype of the H-Store system was shown to significantly outperform a traditional, disk-based RDBMS installation using a well-known OLTP benchmark [10, 11]. The results from this previous work demonstrate that our ideas have merit; we expand on this work and in this paper we present a more full-featured version of the system that is able to execute across multiple machines within a local area cluster. With this new system, we are investigating interesting aspects of this operating environment, including how to exploit non-trivial properties of transactions.

We begin in Section 2 by first providing an overview of the internals of the H-Store system. The justification for many of the design decisions for H-Store is found in earlier work [10]. In Section 3 we then expand our previous discussion on the kinds of transactions that are found in OLTP systems and the desirable properties of an H-Store database design. We conclude the paper in Section 4 by describing the demonstration system that we developed to explore these properties further and observe the execution behavior of our system.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand

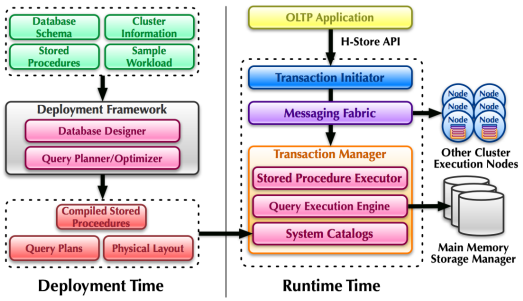

Figure 1: H-Store system architecture.

2. SYSTEM OVERVIEW

The H-Store system is a highly distributed, row-store-based relational database that runs on a cluster on shared-nothing, main memory executor nodes.

We define a single H-Store instance as a *cluster* of two or more computational *nodes* deployed within the same administrative domain. A *node* is a single physical computer system that hosts one or more *sites*. A *site* is the basic operational entity in the system; it is a single-threaded daemon that an external OLTP application connects to in order to execute a transaction. We assume that the typical H-Store node has multiple processors and that each site is assigned to execute on exactly one processor core on a node. Each site is independent from all other sites, and thus does not share any data structures or memory with collocated sites running on the same machine.

Every relation in the database is divided into one or more *partitions*. A partition is replicated and hosted on a multiple sites, forming a *replica set*.

OLTP applications make calls to the H-Store system to repeatedly execute pre-defined *stored procedures*. Each procedure is identified by a unique name and consists of structured control code intermixed with parameterized SQL commands. An instance of a store procedure initiated by an OLTP application is called a *transaction*.

Using these definitions, we now describe the deployment and execution schemes of the H-Store system (see Figure 1).

2.1 System Deployment

H-Store provides an administrator with a cluster deployment framework that takes as input a set of stored procedures, a database schema, a sample workload, and a set of available sites in the cluster. The framework outputs a list of unique invocation handlers that are used to reference the procedures at runtime. It is possible to introduce new procedures into the system after the cluster is deployed; this, however, results in poorer performance as the database's physical design will not be optimized for procedures that are not known at deployment (see Section 3.2).

Internally the deployment framework generates a set of compiled stored procedures and a physical database layout used to instantiate a new H-Store cluster. We currently assume that the administrator provides the framework with a pre-planned database layout. Future versions of the framework will include an automatic database designer that generates an optimized layout using strategic horizontal partitioning, replication locations, and indexed fields.

A compiled stored procedure in the system is comprised of multiple distributed skeleton plans, one for each query in the procedure. Each plan's sub-operations are not specific to database's distributed layout, but are optimized using common techniques found in non-distributed databases. Once this first optimization phase is complete, the compiled procedures are transmitted to each site in the local cluster. At runtime, the query planner on each site converts skeleton plans for the queries into multi-site, distributed query plans. This two-phase optimization approach is similar to strategies used in previous systems [3]. Our optimizer uses a preliminary cost model based on the number of network communications that are needed to process the request. This has proven to be sufficient in our initial tests as transaction throughput is not affected by disk access.

2.2 Runtime Model

At execution time, the client OLTP application requests a new transaction using the stored procedure handlers generated during deployment process. If a transaction requires parameter values as input, the client must also pass these to the system along with the request. We assume for now that all sites are trusted and reside within the same administrative domain, and thus we trust any network communication that the system receives.

Any site in an H-Store cluster is able to execute any OLTP application request, regardless of whether the data needed by a particular transaction is located on that site. Unless all of the queries' parameter values in a transaction are known at execution time, we use a lazy-planning strategy for creating optimized distributed execution plans. If all of the parameter values are known in the beginning, then we perform additional optimizations based on certain transaction properties (see Section 3.1).

When a running transaction executes a SQL command, it makes an internal request through an H-Store API. At this point, all of the parameter values for that query are known by the system, and thus the plan is annotated with the locations of the target sites for each query sub-operation. The updated plan is then passed to a transaction manager that is responsible for coordinating access with the other sites. The manager transmits plan fragments to the appropriate site over sockets and the intermediate results (if any) are passed back to the initiating site. H-Store uses a generic distributed transaction framework to ensure serializability. We are currently comparing protocols that are based on both optimistic and pessimistic concurrency control [5, 12]. The final results for a query are passed back to the blocked transaction process, which will abort the transaction, execute more queries, or commit the transaction and return results back to the client application.

The data storage backend for H-Store is managed by a single-threaded execution engine that resides underneath the transaction manager. Each individual site executes an autonomous instance of the storage engine with a fixed amount of memory allocated from its host machine. Multi-site nodes do not share any data structures with collocated sites, and thus there is no need to use concurrent data structures.

3. DATABASE PROPERTIES

Just as with disk-oriented databases, the performance of distributed main memory databases is dependent on the data placement strategies and supporting data structures of the physical database design. But because OLTP sys-

tems repeatedly execute the same set of transactions, the design can be optimized to execute just these transactions while completely ignoring ad hoc queries. H-Store supports the execution of the latter, but it provides no guarantees that the queries are executed in a timely manner. With this in mind, in the following section we discuss different classes of transactions for our system model; these classes have exploitable properties that enable us to execute them more efficiently than general transactions. We then describe the challenges in designing a database for H-Store that facilitate these classes.

3.1 Transaction Classes

We described in our previous paper a set of special case transaction classes that are found in constrained tree table schemas for OLTP applications [10]. We now provide a more concise description of two of these classes and briefly discuss how H-Store generates an optimal execution strategy for each. Because a query can have unknown parameter values at deployment, the distributed query planner cannot always determine the transaction class before execution time. We plan to investigate the usefulness of probabilistic optimization techniques that estimate the values of these parameters and infer the target site using the cardinality of the columns referenced in the query.

Single-sited Transactions: A transaction is single-sited if all of the queries contained within it can be executed on just one computational site in the cluster. Although the transaction is sent to two or more sites for execution (due to replicated data), it does not require intermediate results from any other site. This transaction class is the most desirable since it does not need to use undo logs, concurrency control mechanisms, or communication with other sites, beyond that which is required for replication.

When H-Store receives a single-sited transaction request, the site that receives the request will transfer execution control along with any compiled plans over to one of the target sites. This new site begins processing the transaction and executes the queries at that site instead of the site where the request was originally made. The overhead from transferring a request between two sites is offset by the performance gains made from avoiding cross-site network communication.

One-shot Transactions: A transaction is one-shot if it cannot execute on a single-site, but each of its individual queries will execute on just one site (not necessarily all the same site). Unlike single-sited transactions, we assume that the outputs of queries in one-shot transactions are not used as input to subsequent queries in the same transaction. H-Store performs a simple optimization technique if a transaction is declared one-shot: the request is transferred to the most accessed site for that transaction instance in the same manner as is done with single-sited transactions.

3.2 Physical Layout

There are two conflicting goals in any H-Store database design: the ability to execute transactions as quickly as possible versus the need to maintain system availability. To achieve the former, a transaction invocation needs to execute on as few sites as possible, thereby minimizing the amount of cross-site messaging between transaction managers. Obvious strategies include replicating frequently accessed or read-only tables on all sites and collocating tables that are joined often together on the same site. More complex designs can involve horizontal partitioning schemes where related data partitions are collocated together or that partition tables more than once using multiple attributes. The problem of choosing a layout scheme becomes exceedingly difficult when deciding which of these policies should take precedence and which procedures to optimize over others.

Furthermore, because H-Store does not keep any data in non-volatile storage, it must rely on strategic data distribution schemes in order to maintain high-availability. Previous literature introduced the concept of *k-safety* in distributed database systems, where k is defined as the number of node failures a database is designed to tolerate [7]. If k is set too low for an H-Store deployment, than a small number of node failures will render the entire system inoperable. Conversely, if k is too high, then the nodes may not have enough memory to store their copies of the data and can reduce the overall system throughput.

Because we will need to generate intelligent layouts for arbitrary schemas and workloads, the development of an automatic database designer is an area of current research.

4. DEMONSTRATION

To illustrate how the database properties discussed in the previous section affects the performance of a distributed OLTP system, we will demonstrate the H-Store system executing on a cluster. We use the TPC-C benchmark's set of transactions, database schema, and pre-generated data as our target domain [11]. When the demonstration system is started, the benchmark workload application connects to each of the sites and begins making transaction execution requests to the H-Store system. The frequency of each transaction is defined by the benchmark weights, but the distribution of the requests made to the sites is random. The demonstration executes in two distinct operational modes: (1) a pre-generated workload with fixed parameter values or (2) a randomly generated workload mode with randomized values. The pre-generated workloads allow the user to directly compare the performance results of different layout schemes. The random mode records the workload as it is generated on-the-fly for future playback. It also allows the user to modify the workload characteristics, such as the frequency that a site receives transaction requests.

4.1 Workload Controller

We provide a graphical interface on a separate machine that allows users to dynamically observe the execution and performance details of a running H-Store system. As shown in the mock-up in Figure 2, this interface provides a heads-up display on the internal message passing and query plan execution in the cluster. The user is allowed to throttle the operating speed of the entire H-Store system and to manually execute a specific stored procedure in isolation.

The workload controller also provides a historical view of the number of transactions executed per second. This timeline view shows aggregate statistics for each operation performed in the system, such as number of rows examined, total number of network messages, and the size of those messages. The user switches the scope of this view based on stored procedure invocation handlers or query operation types in order to view the performance of the system components. This enables a direct comparison of different layout

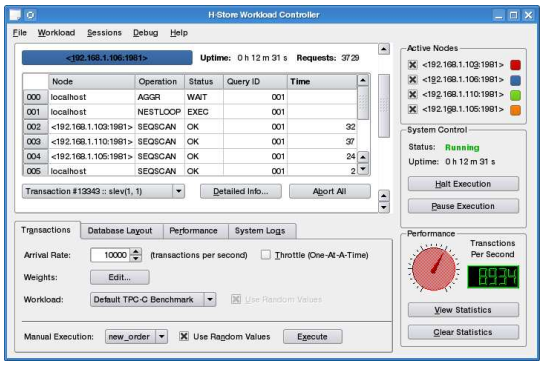

Figure 2: The graphical interface of the H-Store controller allows users to dynamically change properties of the running sample application and change the physical layout of the database.

schemes and workload generation parameters.

4.2 Database Layout Loader

The main focus of our demonstration is to show the impact of differing physical layouts for an H-Store database, and how these layouts affect the execution properties of transactions. Every site in the demonstration cluster is loaded with a set of pre-planned database designs that allow users to observe the implications of various layout schemes. The quality of a particular database layout is based on the number of transactions per second that the cluster can execute.

For each of the three attributes listed below, we generate a separate database design for TPC-C's nine table schema using a manual designer tool. These three designs represent the optimal layout for their corresponding attributes. We also generate a fourth "control" layout where each table is replicated in a random manner on two sites. Each database design only uses a single B-Tree index for each table based on that table's primary key and replicated sets that are sorted on the same column. We ensure that each database layout has a base failure tolerance of one node (k-safety = 1). Although placing tables joined by a foreign key together on the same site improves execution performance, we ignore this optimization strategy in our demonstration.

Table Replication: Every read-only table is replicated on all sites. This design highlights the execution speed gained from avoiding extraneous network communication, thereby shortening the execution time of transactions.

Data Partitioning: All tables are divided horizontally into four distinct partitions that are stored on two separate sites. As opposed to the previous table replication layout scheme, which promotes shorter transaction execution times, this layout demonstrates how concurrent transactions execute in parallel if the data is sufficiently partitioned.

K-Safety: This layout is generated using a horizontal partitioning scheme with a k-safety factor of two. It is not possible to replicate all of the tables in the database on any one site, since the total size of the database is larger than the available memory on a node. We therefore use a replication factor of two and partition each table into equal parts based on its primary key. Every site is given a unique set of three partitions per table, thus preventing any pair of two sites from holding the only copies of a partition.

At any time during workload execution, the user is able to halt the running system, flush the database, and re-load a different layout scheme. Every reload of the database is considered a new workload session, and the system keeps track of the execution results of each session for later review.

5. REFERENCES

[1] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson. System R: relational approach to database management. *ACM Trans. Database Syst.*, 1(2):97–137, 1976.

[2] D. J. DeWitt, R. H. Gerber, G. Graefe, M. L. Heytens, K. B. Kumar, and M. Muralikrishna. Gamma - a high performance dataflow database machine. In *VLDB '86*, pages 228–237, 1986.

[3] W. Hong and M. Stonebraker. Optimization of parallel query execution plans in xprs. In *PDIS '91*, pages 218–225, 1991.

[4] H. V. Jagadish, D. F. Lieuwen, R. Rastogi, A. Silberschatz, and S. Sudarshan. Dali: A high performance main memory storage manager. In *VLDB '94*, pages 48–59, 1994.

[5] I. Lee and H. Y. Yeom. A single phase distributed commit protocol for main memory database systems. In *IPDPS '02*, page 44, 2002.

[6] A.-P. Liedes and A. Wolski. SIREN: A memory-conserving, snapshot-consistent checkpoint algorithm for in-memory databases. In *ICDE '06*, page 99, 2006.

[7] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-Store: A column-oriented dbms. In *VLDB '05*, pages 553–564, 2005.

[8] M. Stonebraker, C. Bear, U. Cetintemel, M. Cherniack, T. Ge, N. Hachem, S. Harizopoulos, J. Lifter, J. Rogers, and S. B. Zdonik. One size fits all? part 2: Benchmarking studies. In *CIDR '07*, pages 173–184, 2007.

[9] M. Stonebraker and U. Cetintemel. "one size fits all": An idea whose time has come and gone. In *ICDE '05*, pages 2–11, 2005.

[10] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *VLDB '07*, pages 1150–1160, 2007.

[11] The Transaction Processing Council. TPC-C Benchmark (Revision 5.9.0). `http://www.tpc.org/tpcc/spec/tpcc_current.pdf`, June 2007.

[12] A. Whitney, D. Shasha, and S. Apter. High volume transaction processing without concurrency control, two phase commit, sql or c. In *Int. Workshop on High Performance Transaction Systems*.