Toward Scalable Transaction Processing

Evolution of Shore-MT

Anastasia Ailamaki Ryan Johnson[†] Ippokratis Pandis[*] Pınar Tözün

[†]University of Toronto
[*]IBM Almaden Research Center
École Polytechnique Fédérale de Lausanne

ABSTRACT

Designing scalable transaction processing systems on modern multicore hardware has been a challenge for almost a decade. The typical characteristics of transaction processing workloads lead to a high degree of unbounded communication on multicores for conventional system designs.

In this tutorial, we initially present a systematic way of eliminating scalability bottlenecks of a transaction processing system, which is based on minimizing unbounded communication. Then, we show several techniques that apply the presented methodology to minimize logging, locking, latching etc. related bottlenecks of transaction processing systems. In parallel, we demonstrate the internals of the Shore-MT storage manager and how they have evolved over the years in terms of scalability on multicore hardware through such techniques. We also teach how to use Shore-MT with the various design options it offers through its sophisticated application layer Shore-Kits and simple Metadata Frontend.

1. INTRODUCTION

In step with Moore's Law, hardware gives us more and more opportunities for parallelism rather than faster processors over the recent years. Exploiting this parallelism is crucial to utilize the available architectural resources and enable faster software. However, designing scalable systems that can take advantage of the underlying parallelism remains as a challenging task for the software developers from various fields.

Transaction processing systems exhibit high concurrency, and therefore, offer a good opportunity for more parallelism. However, the inherent communication in traditional high performance transaction processing systems lead to scalability bottlenecks on today's multicore hardware. Increased hardware parallelism does not automatically bring increased performance for transaction processing. Even systems that are able to scale very well on one generation of multicores might fail to scale-up on the next generation [19].

In this two hour tutorial, We initially teach a clear methodology for scaling-up transaction processing systems on multicore hardware. More specifically, we classify three types of communication in a typical transaction processing system: *unbounded*, *fixed*, and *cooperative* [10]. We demonstrate that the key to achieve scalability on modern hardware, especially for transaction processing systems but also for any system that has similar communication patterns, depends on avoiding the *unbounded* communication points or downgrading them into *fixed* or *cooperative* ones.

Then, we show how effective our methodology is in practice for scaling-up transaction processing systems on multicore hardware. We give examples of some techniques that remove the *unbounded* communication step by step while solving the problem of locking [9, 19], logging [12, 13], and latching [20, 26] bottlenecks in a traditional transaction processing system. We observe how the Shore-MT storage manager [11, 23, 22] has evolved over the years through applying such techniques. We present the implementation of some of these techniques within Shore-MT, illustrating its internals in parallel.

Finally, we introduce the two powerful application layers of Shore-MT, Shore-Kits and Metadata Frontend. Shore-Kits [22] has the implementation of various database workloads for Shore-MT as well as the interface to run Shore-MT using its various design options, while the Metadata Frontend enables creating database tables interactively and running basic ad-hoc queries and transactions on top of Shore-MT.

The goal of this tutorial is to give guidelines for building scalable transaction processing systems by eliminating not all but only the unscalable communication in the system. In addition, it teaches how to use a state-of-the-art open-source scalable storage manager, Shore-MT, as a test-bed for future research. The basic methodology introduced here, however, can be applied to any software system that aims scalability on modern multicore hardware.

Next we describe each part of our tutorial in detail.

2. ELIMINATING THE UNBOUNDED

The communication patterns usually require some form of synchronization and serial execution that eventually limits scalability. Shared-nothing approaches [7, 25] sidestep the issue by disavowing nearly all communication and forbidding any form of tight coupling, while shared-everything systems suffer from communication bottlenecks that limit

their scalability [11]. Since a transaction processing system cannot always eliminate communication without giving up important features, we must find ways to achieve scalability while still allowing some communication.

We classify the communication patterns of a traditional transaction processing system into three types: *unbounded*, *fixed*, and *cooperative*.

Unbounded communication: This type of pattern arises when the number of threads involved with a point of communication is roughly proportional to the degree of parallelism in the system. No matter how efficient or infrequent the communication, exponentially-increasing parallelism will eventually expose it as a bottleneck. Globally shared data structures, which multiple agents update concurrently, fall directly into this category. Used carelessly, unbounded communication can easily dominate execution.

Fixed communication: At the other extreme of the spectrum, fixed communication patterns involve a constant or near-constant number of threads regardless of the degree of parallelism. The pattern itself limits the amount of contention that can arise. Grid-based simulations in scientific computing (including several from the SPLASH-2 benchmark suite [27]) exemplify this type of communication, with each simulated object communicating only with its nearest neighbors in the grid. Peer-to-peer networks (e.g. [24]) employ near-fixed communication patterns as well. Producer-consumer patterns, frequently arising in transaction processing applications, also exhibit fixed communication.

Cooperative communication: A third kind of communication pattern, which we call cooperative, arises when threads work together to reduce contention while waiting to access a shared resource. A canonical example of cooperative communication arises with a parallel LIFO queue: pairs of push and pop requests that encounter each other in flight can cooperate, eliminating themselves directly without further need to compete for the underlying data structure [16]. Such communication is self-moderating: contention increases as more threads communicate, allowing more opportunities for cooperation that in turn reduces contention. Other examples include combining trees and other distributed algorithms.

Examining the three types of communicationsuggests that unbounded communication is the main threat to scalability. The other two types avoid unbounded contention, and increased parallelism usually compensates for any loss of single-thread performance. On the other hand, the impact of communication on the performance and scalability of a system depends on the communication pattern (i.e. how many threads participate), and also on the communications duration. Critical sections implement most communication in shared-memory systems, so the length of each critical section impacts performance strongly. Many unbounded communication or critical sections are short in comparison with others, and they do not appear as bottlenecks when profiling the performance under low parallelism. They are difficult to detect, let alone to prevent, but, the contention they inevitably trigger poses a serious threat to scalability. As code bases mature and measure millions of lines of code (like all the modern database engines) it is increasingly difficult to go back and fix such lurking problems. It is tempting to improve the performance of the system by attacking the longer critical sections, regardless of the types of communication they correspond to; though this approach has short term benefits, bottlenecks usually recur. We argue that longer-term scalability requires a focus on eliminating unbounded communication.

This part of the tutorial takes 15 minutes. We first briefly go over related work. Then, we describe each of the communication types in detail. Finally, we layout our methodology for scalability, which requires a thorough understanding of a system's communication patterns and aims to eliminate the unbounded communication.

3. EVOLUTION OF SHORE-MT

Shore-MT [11, 23] is an enhanced version of the SHORE storage manager [5], whose micro-architectural behavior is very close to the commercial systems [3]. Shore-MT adds a multithreaded storage manager kernel to SHORE and is particularly developed to adapt SHORE to multicore era, mainly by focusing on eliminating the scalability bottlenecks when running on multicore hardware.

Today, Shore-MT is one the most scalable open-source shared-everything storage managers within a single database node. It has been used in different research projects as a test-bed both by the team who develops and maintains it [9, 12, 19, 20, 21] and by other well-known teams in the database and computer architecture communities [4, 8]. As a result, it has evolved over the years quite extensively. Various such proposals for improving transaction processing systems are prototyped by extending its internals. It encompasses many complementary design options that tackle the unbounded communication in transaction processing, which mainly stem from *locking*, *logging*, and *latching*.

Locking: Database locking operates at the logical (application) level to enforce isolation and atomicity between transactions. Techniques like Speculative Lock Inheritance (SLI) [9] and data-oriented execution (DORA) [19] aim to reduce contention due to locking. SLI achieves a performance boost by sidestepping some of the unbounded communication associated with the lock manager, but fails to address the remaining (still-unscalable) communication. DORA, in contrast, eliminates nearly all types of locking, replacing both contention and overhead of centralized communication with efficient, fixed communication via message passing.

Logging: Logging is a bastion of centralized communication in the transaction processing system. Most implementations use some variant of ARIES [14, 15], a write-ahead logging (WAL) scheme that guarantees recoverability in spite of repeated failures, as well as moving a significant fraction of disk I/O off the critical path through asynchronous page cleaning. Unfortunately, WAL requires every update transaction in the system to communicate with the log manager at least twice: one or more times to write-ahead changes, and a final time to request a commit. Techniques like Aether [12, 13], however, manage to consolidate log request and downgrade the unbounded communication due to logging into a cooperative one.

Latching: Page latching operates at the physical (database page) level to enforce the consistency of the physical data stored on disk in the face of concurrent updates from multiple transactions. Eliminating the unbounded communication in *locking* and *logging*, surfaces the unscalable page *latching* as the next scalability bottleneck. Extension of DORA to the physical layers of the system through physiological partitioning (PLP) [20, 26], can minimize the com-

munication due to *latching*, by ensuring single-threaded access to most data structures.

This part of the tutorial is an hour. We illustrate the internals of Shore-MT storage manager and also the code parts from the different design options that implement the various scalability improvements mentioned above.

4. SHORE-KITS

In order to study the behavior and challenges the standardized OLTP benchmarks pose on modern storage managers, we implement them on top of Shore-MT and distribute them as a suite of database benchmarks, called Shore-Kits. In other words, Shore-Kits [22] is an open-source suite of OLTP benchmarks for the Shore-MT storage manager.

Shore-MT does not have an SQL front end, a query parser, and an optimizer. Shore-Kits has the implementation of the benchmarks in C++ using direct calls to Shore-MT's storage manager API, which is linked as a static library to the executable. The transaction or query plans as well as some of the index decisions are taken from the products of various major database vendors. With some programming effort and code refactoring, one can port Shore-Kits to other storage managers by changing the API calls to match the target storage manager's API.

So far, Shore-Kits has four OLTP (TPC-B, TPC-C, TPC-E [2], and TATP [17]), two OLAP (TPC-H [2] and SSB [18]), and one hybrid (CH-benCHmark [6]) benchmarks. In addition, the major functionality that administers the communication in DORA and PLP transactions is also inside Shore-Kits.

This part of the tutorial is 25 minutes. We first show how to download and install Shore-MT and Shore-Kits. Then, we examine the internals of Shore-Kits. Finally, we run Shore-Kits with its different configuration options to see how to enable/disable the various techniques prototyped inside Shore-MT.

5. METADATA FRONTEND

Despite the powerful API of Shore-Kits while implementing various standardized database benchmarks, it has a lot of overhead if one wants to run simple queries or transactions over a simple database table. Therefore, we have recently developed a metadata frontend for Shore-MT. It can interactively switch between different databases, create or drop tables and indexes in a database, and run ad-hoc requests. Moreover, it provides a brief API to be able to easily wrap it up using tools like SWIG [1]. Thus, it is straightforward to automate the process of running simple micro-benchmarks by writing scripts in different programming languages.

This part of the tutorial is 20 minutes. We initially demonstrate how to use the frontend to perform simple tasks. Then, we show how one can use different scripting languages to automatically create and run micro-benchmarks with this frontend.

6. CONCLUSIONS

We teach a clear methodology to achieve scalable transaction processing on multicores. Our main observation is that not all the communication within a transaction processing system is harmful in terms of scalability. We have to identify the *unbounded* communication patterns and find ways to either completely remove them or turn them into *fixed* or *cooperative* types of communication. Throughout this tutorial, we demonstrate several techniques that apply this methodology to eliminate the scalability bottlenecks (such as locking, logging, and latching) in typical transaction processing systems. In parallel, we present the internals and evolution of the Shore-MT storage manager based on these techniques, and introduce its application layers Shore-Kits and Metadata Frontend.

7. BIOGRAPHY

Anastasia Ailamaki (anastasia.ailamaki@epfl.ch)

Anastasia Ailamaki is a Professor of Computer Sciences at the École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland. Her research interests are in database systems and applications, and in particular (a) in strengthening the interaction between the database software and emerging hardware and I/O devices, and (b) in automating database management to support computationally-demanding and demanding data-intensive scientific applications. She has received a Finmeccanica endowed chair from the Computer Science Department at Carnegie Mellon (2007), a European Young Investigator Award from the European Science Foundation (2007), an Alfred P. Sloan Research Fellowship (2005), eight best-paper awards at top conferences (2001-2012), and an NSF CAREER award (2002). She earned her Ph.D. in Computer Science from the University of Wisconsin-Madison in 2000. She is a senior member of the IEEE and a member of the ACM, and has also been a CRA-W mentor.

Ryan Johnson (ryan.johnson@cs.utoronto.ca)

Ryan Johnson is an Assistant Professor at the University of Toronto specializing in systems aspects of database engines, particularly in the context of modern hardware. He contributed heavily to the initial development and performance tuning of Shore-MT. He graduated with M.S. and PhD degrees in Computer Engineering from Carnegie Mellon University in 2010, after completing a B.S. in Computer Engineering at Brigham Young University in 2004. In addition to his work with database systems, Johnson has interests in computer architecture, operating systems, compilers, and hardware design.

Ippokratis Pandis (ipandis@us.ibm.com)

Ippokratis Pandis is a Research Staff Member (RSM) at IBM Research - Almaden in the Advanced Database Solutions group (K55G). Prior joining IBM, Ippokratis graduated from Carnegie Mellon University where he worked on scalable transaction processing on modern hardware architectures. During the last year of his graduate studies, he was also affiliated with the Data-Intensive Applications and Systems (DIAS) Laboratory of École Polytechnique Fédérale de Lausanne (EPFL). His current research focuses on efficient, scalable data management and he is actively involved in IBM's Blink Ultra (BLU) project.

Pınar Tözün (pinar.tozun@epfl.ch)

Pınar Tözün is a fourth year PhD student at École Polytechnique Fédérale de Lausanne (EPFL) working under supervision of Prof. Anastasia Ailamaki in Data-Intensive Applications and Systems (DIAS) Laboratory. Her research focuses on scalability and efficiency of transaction processing systems on modern hardware. Before starting her PhD, she received her BSc degree in Computer Engineering department of Koç University in 2009 as the top student.

8. REFERENCES

[1] Simplified wrapper and interface generator (SWIG). Available at http://www.swig.org.

[2] TPC transcation processing performance council. Available at http://www.tpc.org/default.asp.

[3] A. Ailamaki, D. J. DeWitt, and M. D. Hill. Data page layouts for relational databases on deep memory hierarchies. *VLDB J.*, 11(3):198–215, 2002.

[4] I. Atta, P. Tözün, A. Ailamaki, and A. Moshovos. SLICC: Self-Assembly of Instruction Cache Collectives for OLTP Workloads. In *MICRO*, pages 188–198, 2012.

[5] M. J. Carey, D. J. DeWitt, M. J. Franklin, N. E. Hall, M. L. McAuliffe, J. F. Naughton, D. T. Schuh, M. H. Solomon, C. K. Tan, O. G. Tsatalos, S. J. White, and M. J. Zwilling. Shoring up persistent applications. In *SIGMOD*, pages 383–394, 1994.

[6] R. Cole, F. Funke, L. Giakoumakis, W. Guy, A. Kemper, S. Krompass, H. Kuno, R. Nambiar, T. Neumann, M. Poess, K.-U. Sattler, M. Seibold, E. Simon, and F. Waas. The mixed workload CH-benCHmark. In *DBTest*, pages 8:1–8:6, 2011.

[7] D. J. Dewitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-i. Hsiao, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering - TKDE*, 2(1):44–62, 1990.

[8] G. Graefe, H. Kimura, and H. Kuno. Foster B-trees. *ACM TODS*, 37(3):17:1–17:29, 2012.

[9] R. Johnson, I. Pandis, and A. Ailamaki. Improving OLTP scalability using speculative lock inheritance. *PVLDB*, 2(1):479–489, 2009.

[10] R. Johnson, I. Pandis, and A. Ailamaki. Eliminating unscalable communication in transaction processing. *VLDB J.*, 2013.

[11] R. Johnson, I. Pandis, N. Hardavellas, A. Ailamaki, and B. Falsafi. Shore-MT: a scalable storage manager for the multicore era. In *EDBT*, pages 24–35, 2009.

[12] R. Johnson, I. Pandis, R. Stoica, M. Athanassoulis, and A. Ailamaki. Aether: a scalable approach to logging. *PVLDB*, 3:681–692, 2010.

[13] R. Johnson, I. Pandis, R. Stoica, M. Athanassoulis, and A. Ailamaki. Scalability of write-ahead logging on multicore and multisocket hardware. *VLDB J.*, 21:239–263, 2012.

[14] C. Mohan. ARIES/KVL: a key-value locking method for concurrency control of multiaction transactions operating on B-tree indexes. In *VLDB*, pages 392–405, 1990.

[15] C. Mohan and F. Levine. ARIES/IM: an efficient and high concurrency index management method using write-ahead logging. In *SIGMOD*, pages 371–380, 1992.

[16] M. Moir, D. Nussbaum, O. Shalev, and N. Shavit. Using elimination to implement scalable and lock-free FIFO queues. In *SPAA*, pages 253–262, 2005.

[17] S. Neuvonen, A. Wolski, M. Manner, and V. Raatikka. Telecom application transaction processing benchmark (TATP), 2009. Available at http://tatpbenchmark.sourceforge.net/.

[18] P. O'Neil, B. O'Neil, and X. Chen. Star schema benchmark (SSB), 2009. Available at http://www.cs.umb.edu/ poneil/StarSchemaB.PDF.

[19] I. Pandis, R. Johnson, N. Hardavellas, and A. Ailamaki. Data-oriented transaction execution. *PVLDB*, 3(1):928–939, 2010.

[20] I. Pandis, P. Tözün, R. Johnson, and A. Ailamaki. PLP: page latch-free shared-everything OLTP. *PVLDB*, 4(10):610–621, 2011.

[21] D. Porobic, I. Pandis, M. Branco, P. Tözün, and A. Ailamaki. OLTP on hardware islands. *PVLDB*, 5(11):1447–1458, 2012.

[22] Shore-MT. Shore-MT and Shore-Kits Code Repositories. Available at https://bitbucket.org/shoremt.

[23] Shore-MT. Shore-MT Official Website. Available at http://diaswww.epfl.ch/shore-mt/.

[24] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[25] 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*, pages 1150–1160, 2007.

[26] P. Tözün, I. Pandis, R. Johnson, and A. Ailamaki. Scalable and dynamically balanced shared-everything OLTP with physiological partitioning. *VLDB J.*, 22(2):151–175, 2013.

[27] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *ISCA*, pages 24–36, 1995.