Modern Main-Memory Database Systems

Per-Åke Larson Justin Levandoski

Microsoft Research
palarson@acm.org, justin.levandoski@microsoft.com

ABSTRACT

This tutorial provides an overview of recent developments in main-memory database systems. With growing memory sizes and memory prices dropping by a factor of 10 every 5 years, data having a "primary home" in memory is now a reality. Main-memory databases eschew many of the traditional architectural tenets of relational database systems that optimized for disk-resident data. Innovative approaches to fundamental issues such as concurrency control and query processing are required to unleash the full performance potential of main-memory databases. The tutorial is focused around design issues and architectural choices that must be made when building a high performance database system optimized for main-memory: data storage and indexing, concurrency control, durability and recovery techniques, query processing and compilation, support for high availability, and ability to support hybrid transactional and analytics workloads. This will be illustrated by example solutions drawn from four state-of-the-art systems: H-Store/VoltDB, Hekaton, HyPeR, and SAP HANA. The tutorial will also cover current and future research trends.

1. OVERVIEW

Over the past several years, prominent research systems such as H-Store [4] and HyPer [5] reinvigorated research into main-memory and multi-core data processing. Most major database vendors now have an in-memory database solution, such as SAP HANA [15] and Microsoft SQL Server Hekaton [2]. In addition, a number of startups such as VoltDB [16] and MemSQL have carved out a niche in the database vendor landscape. This first generation of research and production systems offer a very interesting spectrum of design and implementation choices that achieve high performance on memory-bound data. This tutorial provides an overview of the current state-of the art in main-memory systems.

The organization of the tutorial is as follows. We begin with an overview of the history and trends in main-memory database systems. The bulk of the tutorial covers a number of issues and architectural choices that need to be made when building a memory-optimized database, focusing on the novel published work in each area. We describe these topics below.

Proceedings of the VLDB Endowment, Vol. 9, No. 13

1.1 History and Trends

This portion of the tutorial provides a brief summary of earlier research (pre-2005) on main-memory databases and the trends that led to the current renaissance of main-memory database systems. We cover early research systems such as MM-DBMS from the University of Wisconsin [8] through to the early production systems such as P*Time [1] that are the predecessors to todays main-memory engines.

1.2 Issues and Architectural Trends

This section will briefly introduce the key issues and architectural choices that must be resolved when building a main-memory database system. Each issue will then be covered in much more detail as outlined in the subsections below.

1.2.1 In-Memory Data Storage

This issue deals with how database records are represented in memory. Main-memory databases are not constrained by on-disk formats and are free to store data in the format most amenable to achieve performance and system design goals. One representative trend in this space is to avoid clustering records according a primary index, as often done in disk-based relational systems. For example, Hekaton chooses to store individual records in memory in row-oriented format, where one or more indexes point to the in-memory record. As another example, HANA stores data in both row and columnar format to support both OLTP and OLAP style workloads.

1.2.2 Indexing and Data Structure Design

Modern main-memory systems have brought about a renewed interest in high-performance indexing methods. While cache efficiency (a long-standing research issue) remains important [9], multi-core scalability and NUMA-awareness have become equally important. Today, main-memory databases run on CPUs with a staggering amount of parallelism, with each generation of processor increasing the core count even more. Attention to parallelism is especially important for OLTP systems, since indexing is the hot path for updates and retrieval.

In fact, multi-core scalability extends beyond indexing to all critical data structures in the system (e.g., transaction table, durability logging, etc.). As we will see, main-memory systems address multi-core scalability in several ways. Some partition the system (assigning a thread per partition) [4], while others use highly concurrent "latch-free" data structures [10].

1.2.3 Concurrency Control

Main-memory databases running on modern hardware must support a high level of real concurrency while maintaining transactional consistency. Many systems aim to achieve order-of-magnitude

throughput and latency improvements over disk-based architectures. This is accomplished by using new concurrency control approaches. Most modern systems no longer rely on "pessimistic" two-phase locking that blocks threads on conflict. Instead, a number of new approaches have been proposed to unlock the high thread-level parallelism available in todays CPUs. Ideally, these techniques will never block readers, and not starve long read transactions, while still supporting all of the ANSI isolation levels. Modern main-memory systems vary greatly in their concurrency control techniques. Approaches range from partitioning and running transactions serially within a partition [4], to creating new multi-version concurrency control techniques [7, 14].

1.2.4 Durability and Recovery

Volatile RAM is the primary home for data in main-memory systems. However, the database must still provide durability and recovery guarantees in the face of a system shutdowns or crashes (the D in ACID). The use of ARIES-style logging and recovery is rarely used in modern main-memory systems due to performance overheads. They instead use new durability methods that optimize performance on the transactions critical path [2, 11].

1.2.5 Query Processing and Compilation

Since main-memory systems no longer suffer from long I/O bottlenecks on the main processing path, traditional query processing techniques such as interpretation and the iterator model become large overheads. Ideally, query processing in main-memory systems avoid these overheads altogether. Since data is memory-resident, query processing must avoid as much indirection as possible. Several modern main-memory systems achieve performance by compiling queries and stored procedures directly into machine code [3, 13].

1.2.6 Supporting Real-time Analytics

Modern business intelligence pipelines aim to perform OLAP-style queries on a real-time view of operational (OLTP-style) data. With main-memory databases becoming the de facto operational data stores for processing OLTP workloads, it is important that they fit into the real-time analytics pipeline. In general, modern main-memory systems support real-time analytics in two forms. Systems such as HyPeR and HANA support both OLTP and OLAP-style workloads in a single engine. Other systems like Hekaton implement a high-performance pipeline to an analytics engine (e.g., a column-store engine) [6].

1.2.7 High Availability

Many main-memory systems are used in high performance mission critical situations. This calls for support for high availability and failover to hot standbys. Most of the systems we cover support high availability in some form. For instance, H-Store/VoltDB naturally replicates its data to K different servers to achieve k-safety [16]. Hekaton relies on SQL Servers AlwaysOn high availability solution to ship its redo log to secondary failover servers [2].

1.2.8 Clustering and Distribution

A major architectural choice is whether the system will run in a distributed manner or not. Systems like H-Store/VoltDB are built from the ground up to run on a cluster of shared-nothing machines, and handle more load by scaling out. Meanwhile, systems like HyPeR, Hekaton, and HANA are built to run on single machines, and scale up to larger multi-socket machines with large main memories (currently Terabyte in size), with HyPer recently providing scale-out on the OLAP side [12]. There is no right answer for this architectural choice, but this choice affects many of the approaches to achieving high performance, from query processing to concurrency control.

1.3 Active Research and Future Directions

The tutorial concludes with an overview of current and future research directions. Topics in this portion will cover cold data management (what to do with infrequently accessed data), use of non-volatile RAM, hardware transactional memory, and hardware acceleration.

2. REFERENCES

[1] S. K. Cha and C. Song. P*TIME: Highly Scalable OLTP DBMS for Managing Update-Intensive Stream Workload. In *VLDB*, pages 1033–1044, 2004.

[2] C. Diaconu et al. Hekaton: SQL Server's Memory-Optimized OLTP Engine. In *SIGMOD*, pages 1243–1254, 2013.

[3] C. Freedman, E. Ismert, and P. Larson. Compilation in the Microsoft SQL Server Hekaton Engine. *IEEE Data Engineering Bulletin*, 37(1):22–30, 2014.

[4] R. Kallman et al. H-store: A high-performance, distributed main memory transaction processing system. *PVLDB*, 1(2):1496–1499, 2008.

[5] A. Kemper and T. Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *ICDE*, pages 195–206, 2011.

[6] P. Larson, A. Birka, E. N. Hanson, W. Huang, M. Nowakiewicz, and V. Papadimos. Real-Time Analytical Processing with SQL Server. *PVLDB*, 8(12):1740–1751, 2015.

[7] P. Larson, S. Blanas, C. Diaconu, C. Freedman, J. M. Patel, and M. Zwilling. High-Performance Concurrency Control Mechanisms for Main-Memory Databases. *PVLDB*, 5(4):298–309, 2011.

[8] T. J. Lehman and M. J. Carey. Query Processing in Main Memory Database Management Systems. In *ICDE*, pages 239–250, 1986.

[9] V. Leis, A. Kemper, and T. Neumann. The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases. In *ICDE*, pages 38–49, 2013.

[10] J. Levandoski, D. B. Lomet, and S. Sengupta. The Bw-Tree: A B-tree for New Hardware Platforms. In *ICDE*, pages 302–313, 2013.

[11] N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker. Rethinking Main Memory OLTP Recovery. In *ICDE*, pages 604–615, 2014.

[12] T. Mühlbauer, W. Rödiger, A. Reiser, A. Kemper, and T. Neumann. ScyPer: Elastic OLAP Throughput on Transactional Data. In *DanaC*, 2013.

[13] T. Neumann. Efficiently Compiling Efficient Query Plans for Modern Hardware. *PVLDB*, 4(9):539–550, 2011.

[14] T. Neumann, T. Mühlbauer, and A. Kemper. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. In *SIGMOD*, pages 677–689, 2015.

[15] V. Sikka, F. Färber, W. Lehner, S. K. Cha, T. Peh, and C. Bornhövd. Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth. In *SIGMOD*, pages 731–742, 2012.

[16] M. Stonebraker and A. Weisberg. The VoltDB Main Memory DBMS. *IEEE Data Engineering Bulletin*, 36(2):21–27, 2013.