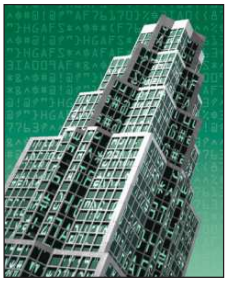
Big Data

Editor: Tim Kraska, tim_kraska@brown.edu



Emerging Hardware Trends in Large-Scale Transaction Processing

Andrew Pavlo • *Carnegie Mellon University*

What are the next challenges and trends for online transaction-processing (OLTP) database systems? Here, Andrew Pavlo opines on how database management system developers will need to respond as new hardware technologies become widely available.

New ideas for research always come when you least expect it. In my case, it was a dreary Saturday morning with my research mentor Leon Wrinkles. The party was over, the cops had left, and we were left with two dead cats that we had to dispose of before Leon's lover returned from her trip to the sanitarium.

As we dug the grave at the local playground, Leon posed a question to me that turned out to be prolific: What are the next challenges for new hardware for transaction processing database systems?

It's rare that you're able to recognize a turning point in your life when it occurs. As a new professor still getting settled at Carnegie Mellon, I hadn't had the time to really think about where research is heading beyond the next one or two years. But he was right; I needed to look further into the future.

Thus, the following are my thoughts about where there are still interesting problems in online transaction-processing (OLTP) database systems for emerging hardware.

Non-Volatile Memory

Back in 2007, H-Store was the vanguard for a new era of main memory-oriented database management systems (DBMSs).[1] It wasn't the first transactional, in-memory DBMS (TimesTen and DataBlitz are earlier examples from the 1990s), but it was one of the first systems created after the NoSQL crowd started bleating that the only way to scale a database system is to give up ACID (atomicity, consistency, isolation, and durability).

It wasn't obvious right away to some that a memory-oriented architecture was the way to go for scalable OLTP applications. In the early days of VoltDB (the commercial implementation of H-Store), customers were uncomfortable with the idea of storing your entire database in volatile DRAM. I visited PayPal with others in 2009 to talk to them about VoltDB, and I remember that their senior management was unnerved by the idea of a DBMS that didn't store all physical changes to tuples immediately on disk. The prevailing conventional wisdom has obviously changed, and now many mission-critical applications use VoltDB. Since then, several other memory-oriented systems are now available, including MemSQL, SAP HANA,[2] and Microsoft Hekaton.[3] Other notable in-memory academic systems that came along after H- Store include Shore-MT,[4] HyPer,[5] and Silo.[6]

In my opinion, the big challenges in main memory DBMSs are mostly solved. There are, of course, engineering problems that must be solved for customers in the commercial world. But the next systems that researchers are going to build won't focus on fast transaction processing in a DRAM-based DBMS using today's CPU architectures. A more interesting topic is how the advent of non-volatile memory (NVM) will overturn the traditional storage hierarchy in computing systems. DBMSs are uniquely positioned to use this technology for a variety of application domains.

The first incarnations of NVM will be block-addressable PCI-e cards. Based on my discussions with hardware vendors, these devices are

Published by the IEEE Computer Society
1089-7801/15/$31.00 © 2015 IEEE

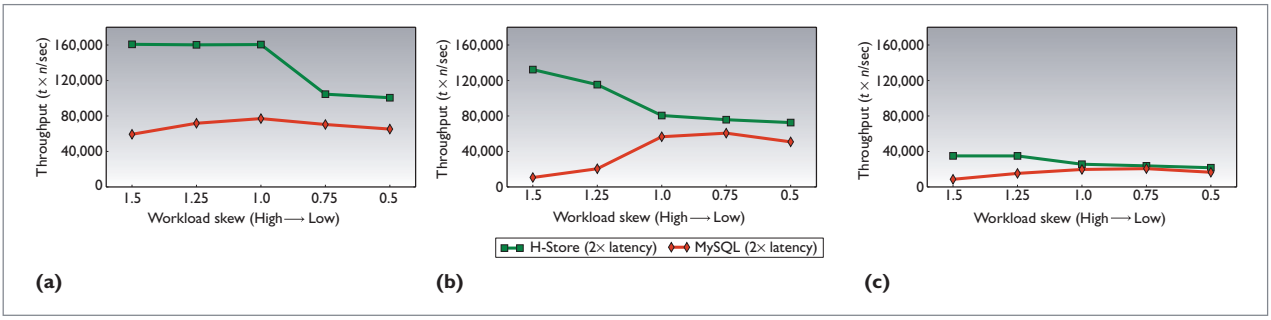


Figure 1. H-Store and MySQL running on Intel Lab's non-volatile memory (NVM) hardware emulator with an ×320-nanosecond read/write latency (2× the speed of DRAM). (a) Read-only, (b) read-heavy, and (c) write-heavy workloads.[9]

less than three years away. Memory-oriented DBMSs will still be the best-performing system architecture for this hierarchy, because they don't use legacy architectural components from the 1970s designed to mask the latency of slow disks.[7]

Any DBMS that still uses DRAM for ephemeral storage will need to flush out changes to stable storage for recovery and durability. This operation will be the major bottleneck for all DBMSs, even if it has fast NVM. We did some initial experiments testing MySQL and H-Store on Intel Lab's NVM emulator[8] as part of the Intel Science and Technology Center (ISTC) for Big Data. We configured the emulator so that all reads/writes to NVM were approximately 320 nanoseconds (ns) compared to a roughly 160-ns read/write to DRAM. We used three workload variants of the Yahoo Cloud Serving Benchmark (YCSB): *read-only*, *read-heavy*, and *write-heavy*.[9]

As Figure 1 shows, our experiments indicated a significant decrease in throughput as the number of update transactions in the workload increases. H-Store exhibits a 77 percent drop in peak performance for the read-only workload compared to its best performance in the write-heavy workload. For MySQL, this difference is nearly 75 percent. This is due to the overhead of preparing and writing the log records out to durable storage to overcome DRAM's volatility. We also see that for the read-heavy workload, H-Store achieves 13× better throughput over MySQL when skew is high, but only a 1.3× improvement when skew is low. This is because H-Store performs best when there's a high skew, since it needs to fetch fewer blocks from the NVM anti-cache and restart fewer transactions. In contrast, the disk-oriented system performs worse on the highly skewed workloads due to lock contention. But this performance difference is nearly non-existent for the write-heavy workload. We attribute this to the overhead of logging that's inherent in legacy systems.

Rather than trying to retrofit an existing system to fix these problems, we're developing a new DBMS at CMU, designed specifically for NVM. Our goal is to develop the fundamental principles of how to use byte-addressable NVM to support data-intensive applications in a way that isn't possible with today's DBMSs.[10] The culmination of this work is embodied in a new open source DBMS that we dubbed *N-Store* (the "non-volatile store"; http://nstore.cs.cmu.edu). N-Store will be a hybrid system (with Hybrid Transactional/Analytical Processing, or HTAP, capabilities) that supports simultaneous high-velocity OLTP transactions with full ACID guarantees and real-time online analytical-processing (OLAP) queries.

I think that hybrid DBMSs are the future of OLTP DBMSs, but they won't completely replace data warehouses (such as Vertica or Impala). These DBMSs ingest data from multiple front-end applications using an extract, transform, and load (ETL) process to convert them into a uniform schema, whereas in a hybrid system the schema doesn't change. But a system like N-Store lets developers implement analytical functionality directly in their OLTP application more easily, rather than it being an afterthought. This will open up many possibilities, because now applications can use powerful analytical operations immediately as data are created. This is better than waiting to transfer the data to the data warehouse and then finding out that there's a key piece of data that they could have collected but didn't.

Many-Core CPU Architectures

In the mid-2000s, the trend in CPU architectures shifted from increasing clock speeds of single-threaded execution to multicore CPUs. Clock frequencies have increased for decades, but now the growth has stopped because of hard power constraints and complexity issues. Aggressive, out-of-order, superscalar processors are being replaced with simple, in-order, single-issue cores.

Much of the research and development in DBMSs for multicore has been on adapting existing architectures that assumed a single CPU core to utilize additional cores. The work on optimizing Shore-MT is probably the best example of this.[4] Although this work was important, timely, and necessary, the number of cores that it targeted was relatively small compared to what we expect future processors to support.

Soon we'll enter the era of "many-core" machines powered by potentially

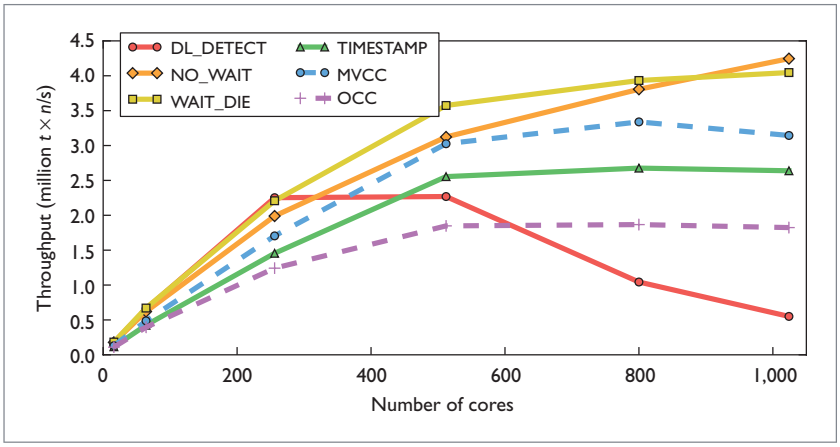


Figure 2. Comparison of the concurrency control schemes with DBx1000 running in Graphite using the Yahoo Cloud Serving Benchmark (YCSB) workload with medium contention.[10]

hundreds to thousands of these smaller, low-power cores on a single chip. The previous research on multicore CPUs was all about software approaches that used hardware in a smarter way. But the difference is that with many-cores there seems to be a fundamental barrier into what gains we can achieve with software. Hence, the advancements from the previous decade for running DBMSs on multicore CPUs won't be enough when the core counts are much greater.

In an ongoing research project at MIT and CMU, we're trying to determine whether this assumption holds for OLTP DBMSs running on a CPU with 1,000 cores.[11] Such chips obviously don't exist yet, so we've been using a CPU simulator developed at MIT called Graphite[12] and using a new experimental DBMS called DBx1000 to test different aspects of transaction processing (https://github.com/yxymit/DBx1000). Graphite's simulated architecture is a tiled chip multiprocessor where each tile has a small processor core, two levels of cache, and a 2D mesh network-on-chip for communication between the cores. This is similar to other commercial CPUs, such as Tilera's Tile64 (64-core), Intel's SCC (48-core), and Intel's Knights Landing (72-core).

Our initial experiments test what happens when executing transactions with high core counts. We started with concurrency control schemes, as this is always the main bottleneck that you must address first in an OLTP DBMS. Our system supports a pluggable lock manager that lets us swap in different two-phase locking and timestamp-ordering concurrency control schemes. We adapted these schemes from the great Phil Bernstein's seminal work[13] on concurrency control algorithms and implemented state-of-the-art variants. We again compared six of these concurrency control schemes using a write-heavy YCSB workload with skew.[11]

The results in Figure 2 show that the two-phase locking variants NO_WAIT and WAIT_DIE are the only schemes that scale past 512 cores, but even they fail to scale up to 1,000 cores. Many of the schemes are inhibited by lock thrashing and high transaction abort rates. This is because there are too many concurrent threads trying to access the same set of records. Clearly, no scheme is ideal here.

The next question is how can we overcome this performance barrier? Initially, I started drinking heavily to ease the pain of knowing that DBMSs won't be able to scale to 1,000 cores. I always suspected that this was the case, but something snapped in my head once I saw the results and it drove me to look for answers in the bottom of the bottle. This wasn't sustainable (both financially and medically), and thus we need to come up with a way to actually solve this problem.

I believe that the next trend in the many-core era is the development of software-hardware co-design for new DBMS architectures. On the software side, rather than attempting to remove scalability bottlenecks of existing DBMS architectures through incremental improvements, a better approach will be to redesign the system from the bottom-up to target many-core from inception. On the hardware side, instead of simply adding more cores to a single chip, there must be new hardware components included on the CPU itself that can unburden the software system from computationally critical tasks.

The future of leveraging new hardware for transaction processing systems is ripe. You could argue that the emergence of NVM and many-core CPUs doesn't represent a new computing paradigm for transaction-processing DBMSs. Specialized components, such as field-programmable gate arrays (FPGAs) and DRAM backed with supercapacitors, have been around for years, and essentially provide the same benefits. This previous hardware, however, wasn't adopted in commodity systems and thus there are only a small number of DBMSs designed to use them (such as Netezza[14]).

In the near future, NVM and many-core will become standard hardware components, so cloud providers will deploy systems that include them. Thus, DBMS developers will be unable to ignore them and be forced to make changes to their architectures to use them.

References

1. R. Kallman et al., "H-Store: A High-Performance, Distributed Main Memory Transaction Processing System," *Proc. VLDB Endowment*, vol. 1, no. 2, 2008, pp. 1496–1499.

2. F. Färber et al., "SAP HANA Database: Data Management for Modern Business Applications," *SIGMOD Record*, vol. 40, no. 4, 2012, pp. 45–51.
3. C. Diaconu et al., "SQL Server's Memory-Optimized OLTP Engine," *Proc. SIGMOD*, 2013, pp. 1–12.
4. R. Johnson et al., Shore-MT: A Scalable Storage Manager for the Multicore Era," *Proc. Conf. Extending Database Technology*, 2009, pp. 24–35.
5. A. Kemper and T. Neumann, "HyPer: A Hybrid OLTP&OLAP Main Memory Database System Based on Virtual Memory Snapshots," *Proc. ICDE*, 2011, pp. 195–206.
6. S. Tu et al., "Speedy Transactions in Multicore In-Memory Databases," *Proc. 24th ACM Symp. Operating Systems Principles*, 2013, pp. 18–32.
7. S. Harizopoulos et al., "OLTP through the Looking Glass, and What We Found There," *Proc. SIGMOD*, 2008, pp. 981–992.
8. S.R. Dulloor et al., "System Software for Persistent Memory," *Proc. European Conf. Computer Systems*, 2014, article no. 15.
9. J. DeBrabant et al., "A Prolegomenon on OLTP Database Systems for Non-Volatile Memory," *Proc. VLDB Endowment*, vol. 7, no. 14, 2014, pp. 57–63.
10. J. Arulraj, A. Pavlo, and S. Dulloor, "Let's Talk About Storage & Recovery Methods for Non-Volatile Memory Database Systems," *Proc. SIGMOD*, to be published, 2015.
11. X. Yu et al., "Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores," *Proc. VLDB Endowment*, vol. 8, no. 3, 2014, pp. 209–220.
12. J. Miller et al., "Graphite: A Distributed Parallel Simulator for Multicores," *Proc. High-Performance Computer Architecture*, 2010, pp. 1–12.
13. P.A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys*, vol. 13, no. 2, 1981, pp. 185–221.
14. M. Singh and B. Leonhardi, "Introduction to the IBM Netezza Warehouse Appliance," *Proc. Conf. Center for Advanced Studies on Collaborative Research*, 2011, pp. 385–386.

Andrew Pavlo is an assistant professor of databaseology in the Computer Science Department at Carnegie Mellon University. His research focuses on database management systems, specifically main memory systems, non-relational systems (NoSQL), transaction processing systems (NewSQL), and large-scale data analytics. Pavlo has a PhD in computer science from Brown University. Contact him at pavlo@cs.cmu.edu.