Staring into the Abyss:
An Evaluation of Concurrency Control with One Thousand Cores[1]

Xiangyao Yu, George Bezerra, Andrew Pavlo, Srini Devadas, Michael Stonebraker

The era of exponential single-threaded performance improvement is over. Hard power constraints and complexity issues have forced chip designers to move from single- to multi-core designs. Clock frequencies have increased for decades, but now the growth has stopped. Aggressive, out-of-order, super-scalar processors are now being replaced with simple, in-order, single issue cores [1]. We are entering the era of the many-core machines that are powered by a large number of these smaller, low-power cores on a single chip. Given the current power limits and the inefficiency of single-threaded processing, unless a disruptive technology comes along, increasing the number of cores is currently the only way that architects are able to increase computational power. This means that instruction-level parallelism and single-threaded performance will give way to massive thread-level parallelism.

As Moore's law continues, the number of cores on a single chip is expected to keep growing exponentially. Soon we will have hundreds or perhaps a thousand cores on a single chip. The scalability of single-node, shared-memory DBMSs is even more important in the many-core era. But if the current DBMS technology does not adapt to this reality, all this computational power will be wasted on bottlenecks, and the extra cores will be rendered useless.

In this talk, we take a peek at this dire future and examine what happens with transaction processing at one thousand cores. Rather than looking at all possible scalability challenges, we limit our scope to concurrency control. With hundreds of threads running in parallel, the complexity of coordinating competing accesses to data will become a major bottleneck to scalability, and will likely dwindle the gains from increased core counts. Thus, we seek to comprehensively study the scalability of OLTP DBMSs through one of their most important components.

2PL	**DL_DETECT**	2PL with deadlock detection.
	NO_WAIT	2PL with non-waiting deadlock prevention.
	WAIT_DIE	2PL with wait-and-die deadlock prevention.
T/O	**TIMESTAMP**	Basic T/O algorithm.
	MVCC	Multi-version T/O.
	OCC	Optimistic concurrency control.
	H-STORE	T/O with partition-level locking.

Table 1: The con currency control schemes evaluated

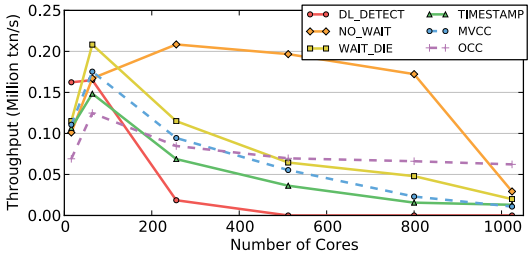

Figure 1: write intensive YCSB with high contention.

We implemented seven concurrency control algorithms in a main memory DBMS and used a high-performance, distributed CPU simulator to scale the system to 1000 cores. Implementing a system from scratch allows us to avoid any artificial bottlenecks in existing DBMSs and instead understand the more fundamental issues in the algorithms. Previous scalability studies used existing DBMSs, but many of the legacy components of these systems do not target many-core CPUs. To the best of our knowledge, there has not been an evaluation of multiple concurrency control algorithms on a single DBMS at such large scale.

Our analysis shows that all algorithms fail to scale as the number of cores increases (Figure 1). In each case, we identify the primary bottlenecks that are independent of the DBMS implementation and argue that even state-of-the-art systems suffer from these limitations. We conclude that to tackle this scalability problem, new concurrency control approaches are needed that are tightly co-designed with many-core architectures. Rather than adding more cores, computer architects will have the responsibility of providing hardware solutions to DBMS bottlenecks that cannot be solved in software. In the following of this abstract, we show two of these bottlenecks as examples.

Bottleneck 1: Lock Thrashing

Lock based concurrency control algorithms do not scale due to lock thrashing. This occurs when a transaction holds its locks until it commits, blocking all the other concurrent transactions that attempt to acquire those locks. As shown in Figure 2, this becomes a problem with high contention and a large number of concurrent transactions, and thus is the main bottleneck of all 2PL schemes.

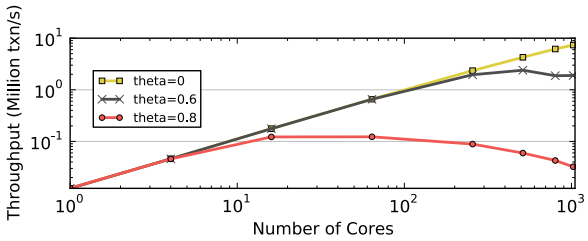

Figure 2. Lock Thrashing – Results for a write-intensive YCSB workload using the DL_DETECT scheme without deadlock detection. Each transaction acquires locks in their primary key order.

Bottleneck 2: Timestamp Allocation

All T/O-based algorithms make ordering decisions based on transactions' assigned timestamps. The DBMS must therefore guarantee that each timestamp is allocated to only one transaction. A naïve approach to ensure this is to use a mutex in the allocator's critical section, but this leads to poor performance. Another common solution is to use an atomic addition operation to advance a global logical timestamp. This requires fewer instructions and thus the DBMS's critical section is locked for a smaller period of time than with a mutex. But as shown in Figure 3, this approach is still insufficient for a 1000-core CPU. We proposed three timestamp allocation alternatives that are more scalable: (1) atomic addition with batching, (2) CPU clocks, and (3) hardware counters.

CPU clocks and hardware counters are the two most scalable solutions. Both require hardware supports. To generate a timestamp using clock-based allocation, each worker thread reads a logical clock from its local core and then concatenates it with its thread id. This provides good scalability as long as all the clocks are synchronized. On a many-core CPU, however, clock synchronization imposes large overhead and thus requires hardware support. As of July 2014, only Intel CPUs support synchronized clocks across cores.

Another approach is to use an efficient, built-in hardware counter. The counter is physically located at the center of the CPU such that the average distance to each cores is minimized. No existing CPU currently supports this.

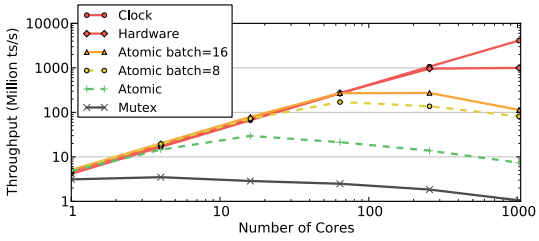

Figure 3. Timestamp Allocation Micro-benchmark – Throughput measurements for different timestamp allocation methods.

[1] Xiangyao Yu, George Bezerra, Andrew Pavlo, Srinivas Devadas, Michael Stronebraker, **"Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores"**, Proceedings of the VLDB Endowment, vol. 8, iss. 3, pages. 209—220, November 2014.