The VLDB Journal
https://doi.org/10.1007/s00778-022-00742-4

SPECIAL ISSUE PAPER

The full story of 1000 cores

An examination of concurrency control on real(ly) large multi-socket hardware

Tiemo Bang[1] · Norman May[2] · Ilia Petrov[3] · Carsten Binnig[4]

Received: 7 May 2021 / Revised: 15 December 2021 / Accepted: 19 March 2022

Abstract
In our initial DaMoN paper, we set out the goal to revisit the results of "Starring into the Abyss [...] of Concurrency Control with [1000] Cores" (Yu in Proc. VLDB Endow 8: 209-220, 2014). Against their assumption, today we do not see single-socket CPUs with 1000 cores. Instead, multi-socket hardware is prevalent today and in fact offers over 1000 cores. Hence, we evaluated concurrency control (CC) schemes on a real (Intel-based) multi-socket platform. To our surprise, we made interesting findings opposing results of the original analysis that we discussed in our initial DaMoN paper. In this paper, we further broaden our analysis, detailing the effect of hardware and workload characteristics via additional real hardware platforms (IBM Power8 and 9) and the full TPC-C transaction mix. Among others, we identified clear connections between the performance of the CC schemes and hardware characteristics, especially concerning NUMA and CPU cache. Overall, we conclude that no CC scheme can efficiently make use of large multi-socket hardware in a robust manner and suggest several directions on how CC schemes and overall OLTP DBMS should evolve in future.

Keywords Databases · Modern hardware · Benchmarking · OLTP · Concurrency control · Multi-socket

1 Introduction

We are now 8 years after "Starring into the Abyss [...] of Concurrency Control with [1000] Cores" [71], which presented an evaluation of concurrency schemes for in-memory databases on simulated hardware. The speculation of the authors at that time was that today we would see single CPUs with 1000s of cores. However, so far reality is different [22,30,31,36]. Instead, multi-socket hardware indeed offering 1000s of cores made their way into production data centres. Accordingly, in-memory DBMS are facing not only challenges of massive thread-level parallelism, such as coordination of hundreds of concurrent transactions as predicted by [71], but multi-socket systems also expose in-memory DBMS to further challenges, such as deep NUMA topologies connecting all CPUs [21,23,30,31].

In this paper, we thus set out the goal to bring in-memory DBMS to 1000 cores on today's multi-socket hardware, revisiting the results of the simulation of [71] based on the original code, which the authors generously provide as open source. That is, we follow-up on [71] with an evaluation of the characteristics of concurrency control (CC) schemes on real production hardware using their DBx1000 as a starting point. As the main contribution, we provide an extensive analysis of CC schemes on real large hardware, which beyond related evaluation works [2,19,48,52,69,71] provides a breadth of insights for OLTP on modern multi-socket hardware platforms, as discussed below. Moreover, as another contribution of this paper we have released all artefacts [5,6] (code and measurements) for further analysis by the database community. While we already provide an extensive analysis of our

✉ Tiemo Bang
tiemo.bang@cs.tu-darmstadt.de

Norman May
norman.may@sap.com

Ilia Petrov
ilia.petrov@reutlingen-university.de

Carsten Binnig
carsten.binnig@cs.tu-darmstadt.de

1 Technical University of Darmstadt & SAP SE, Darmstadt, Germany

2 SAP SE, Walldorf, Germany

3 Reutlingen University, Reutlingen, Germany

4 Technical University of Darmstadt, Darmstadt, Germany

Published online: 29 April 2022

data, we believe that the data itself is an interesting source for future findings simply by analysing the data even further.

1.1 Part one

This part is based on the results of our recent DaMoN paper [3] where we analysed concurrency control schemes on an Intel multi-socket hardware with 1568 cores. To our surprise, we made several interesting findings: (1) Overall, running the DBx1000 open source prototype of [71] on today's production hardware revealed a very different picture compared to prior observations on simulated hardware with 1000 cores. While simulations indeed are valuable for early path finding, today's real hardware and progress of state-of-the-art have changed the prospect for OLTP on 1000 cores. (2) In a "deeper look", we additionally revisited the limitations and assumptions of the simulation for our real hardware. For example, we found that hardware-assisted timestamp allocation indeed available today has ambiguous benefits, as physical contention shifts rather than disappears. Moreover, aspects of real systems (e.g. memory management) have proven significant performance impact apart from concurrency control. (3) Based on these findings, we revised the original prototype for large multi-socket hardware and finally the experimental results gave a clear view on concurrency control with 1000 cores, i.e. good scaling of all CC schemes under low conflict and textbook behaviour under high conflict though with thrashing.

1.2 Part two

This new part extends our DaMoN paper [3] significantly and broadens the evaluation in two dimensions: hardware and workload. First, we additionally include two IBM Power-based platforms (Power8 and Power9), which come with different hardware characteristics on the macro-level (e.g. their overall topology) as well as the micro-level (e.g. their simultaneous multithreading implementation). The focus of this part is on singling out the effects of hardware characteristics on the CC schemes (e.g. of different NUMA topologies, cache capacities). Second, we also extended our evaluation in terms of the workload. While in the first part, we only used the common limited transactions mix of the TPC-C benchmark that was available in DBx1000, in part two we also analyse how the full TPC-C transaction mix effects concurrency control on our large multi-socket hardware. The most compelling findings of our deep dive into hardware and workload characteristics are: (1) We could identify clear connections between the performance of the individual CC schemes and specific hardware characteristics. For example, NUMA had an outstanding but nuanced effect on the CC schemes. (2) The significance of hardware characteristics like NUMA effects further depends on the workload. For example, a larger footprint of the workload (accessed tuples) increases the bandwidth demand of the optimistic concurrency control (OCC) scheme and thus, OCC scales as long as the underlying NUMA architecture offers sufficient bandwidth. (3) Under high conflict, no CC scheme achieves high concurrency on any hardware platform. Our analysis here surfaced inherent issues of today's transaction execution, i.e. increasing inter-transaction parallelism under high conflict will not help without changing the CC schemes and execution schemes.

Overall, our evaluation exhibits the complex interaction of the system design, the workload, and the underlying hardware that determines DBMS performance. Therefore, we recommend reflecting on concurrency in OLTP DBMSs, to put available hardware resources to effective use. Especially with hundreds to thousands of cores, we need broader options for utilising those, apart from executing more concurrent transactions and comprehensive contention management is imperative. Finally, we advocate for performance models to aid exploration of system performance and adaptive DBMS designs towards robust performance in any condition.

1.3 Outline

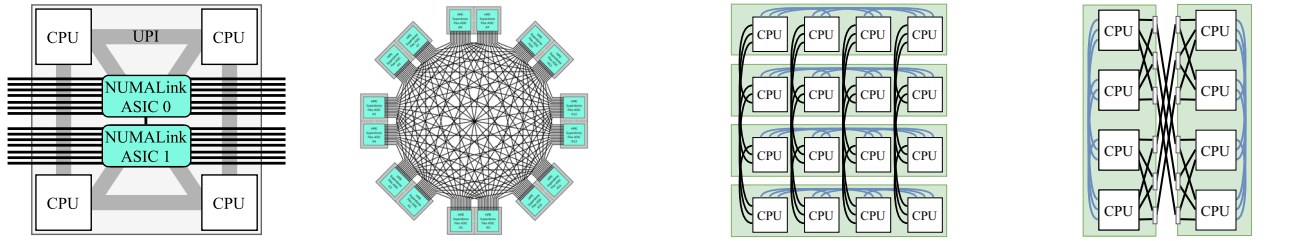
We first provide the background and present the different hardware platforms used in this paper (Sect. 2). In Sect. 3, we then present the results of part one which is based on [3], as mentioned before. Afterwards, in Sect. 4 we present the results of part two which includes the findings of our broadened evaluation with additional hardware platforms and the full TPC-C benchmark. Finally, we conclude with a summary and discussion of the overall findings in Sect. 5.

2 Background and setup

In the following, we provide a brief overview of the concurrency control (CC) schemes, the hardware as well as the benchmarking environment used in our evaluation.

2.1 Concurrency control schemes

Table 1 summarises the evaluated CC schemes. They range from lock-based CC, with diverse mechanisms against deadlocks, to timestamp-ordering-based CC, including multi-versioning, 2-versioning, coarse locking, and advanced ordering. For details on these CC schemes, we refer to their original publications [7,8,32,35,62,73] and to [71]. The first 7 CC schemes in Table 1 correspond to the prior evaluation in [71]. We further include the more recent schemes SILO [62] and TICTOC [73], originally not included. Unfortunately, TIMESTAMP [7] from [71] has a fatal bug in the latest version of the prototype, so we excluded it.



(a) Chassis topology HPE **(b)** NUMALink topology HPE **(c)** 16-socket IBM Power9 **(d)** 8-socket IBM Power8

Fig. 1 System topologies of the HPE, Power9, and Power8 platforms. [21,23,49,63]

Table 1 Evaluated concurrency control schemes

DL DETECT	2PL with deadlock detection [7]
NO WAIT	2PL with non-waiting deadlock prevention [7]
WAIT DIE	2PL with wait-and-die deadlock prevention [7]
MVCC	Multi-version T/O [8]
OCC	Optimistic concurrency control [35]
HSTORE	T/O with partition-level locking [32]
SILO	Epoch-based T/O [62]
TICTOC	Data-driven T/O [73]

2.2 Today's real hardware with 1000 cores

The prevalent hardware in production today offering 1000 cores is large multi-socket hardware platforms [22,30,31,36]. Rather than hosting many cores on a single CPU, these multi-socket platforms connect many CPUs to a single system. In the following, we first introduce the Intel-based HPE platform used in the comparison with the simulation and the later comparison of different real multi-socket platforms. Then, we introduce two further IBM platforms. Notably, we choose these three specific platforms for comparing large multi-socket hardware with diverse characteristics, especially different NUMA topologies. In general, we expect systems with similar NUMA topologies to have similar NUMA effects.

2.2.1 Intel-based HPE platform

Our *HPE SuperdomeFlex* system [22], used for part one of our evaluation (Sect. 3), contains 28 *Intel Xeon 8180* CPUs each having 28 physical cores with SMT-2 [39]. This makes a total of 1568 logical cores (hardware threads), as shown in Table 2a. Figure 1a, b shows how this system groups 4 CPUs into hardware partitions (chassis) [21] and then joins these [23], forming a single cache coherent system with the total of 1568 logical cores and 20 TB of DRAM. As shown in Fig. 1a, within the chassis each CPU connects to two neighbouring CPUs and to a *NUMALink* controller via UPI links. In turn, the *NUMALink* controllers couple all chassis in a fully connected topology (Fig. 1b), yielding 4 levels of NUMA with performance properties summarised in Table 2c[1].

Comparing this hardware to potential many-core hardware as simulated in [71] reveals that this multi-socket setup for 1000 cores differs in many aspects. Importantly, one similarity of today's hardware to the simulated architecture of [71] is that both communicate and share cache in a non-uniform manner via a 2D-mesh on the chip [17] (and UPI beyond), such that the cores use the aggregated capacity to cache data but need to coordinate for coherence. This non-uniform communication is an important hardware characteristic, as it can amplify the impact of contention points in the CC schemes on any large hardware (multi-socket and many-core). Otherwise, the simulation differs from today's hardware, since it assumed low-power and in-order processing cores clocked at 1GHz, cache hierarchies with only two levels, and cache capacities larger than today's caches. Notably, it simulates the DBMS in isolation without an OS, disregarding overheads and potential side effects of OS memory management, scheduling, etc., omitting essential aspects of real systems [15,44].

2.2.2 Power-based IBM platforms

In the second part of our evaluation, we consider the IBM Power platforms as prominent hardware platforms for scale-up systems, in addition to the Intel-based platform. There are several distinctive features, making these IBM Power platforms an interesting alternative for our analysis [30,31,49].

In particular, we use an IBM Power system E880 (Power8) [63] configured with 8 Power8 CPUs and an IBM Power system E980 (Power9) [64] configured with 16 Power9 CPUs. As outlined in Table 2a, these platforms offer a total of 768

[1] NUMAPerf is a cross-platform tool in HCMT [46] that performs similar tests like Intel MLC [65]. Different from MLC it provides comparable results across platforms. While it clearly does not implement platform-specific optimisations and thus observed performance can be
Footnote 1 continued
lower, NUMAPerf allow us to compare the performance across platforms.

Table 2 Properties of evaluated hardware platforms

(a) Number of CPUs, physical cores, and logical cores. In parentheses is the number of logical cores available to the application on Power8 and Power9

Platform	CPUs	∗ Phy. Cores	∗ SMT	= Logical Cores
HPE	28	∗ 28	∗ 2	= 1568
Power9	16	∗ 12	∗ 8	= 1536 (1504)
Power8	8	∗ 12	∗ 8	= 768 (752)

(b) Cache capacity in KB per physical core and in parentheses per logical core. Agg. denotes the aggregated cache capacity of the L2 and the non-inclusive L3 cache. [39,49,50]

Platform	L1 Inst., Data	L2	L3	Agg.
HPE	32, 32 (16, 16)	1024 (512)	1408 (687.5)	2432 (1216)
Power9	64, 64 (4, 4)	512 (64)	10240 (1280)	10752 (1344)
Power8	32, 64 (2, 4)	512 (64)	8192 (1024)	8704 (1088)

(c) Memory access latency and bandwidth by NUMA distance measured with NUMAPerf[1] [46]. HPE has a deeper topology than the Power platforms. Distance 0 (*Local*) refers to memory directly located the CPU, 1-2 (*1 Hop*, *2 Hop*) refer to memory of neighbouring CPUs within the same chassis, and 3 (*Remote*) refers to accesses across chassis.

Platform	NUMA Distance	Latency(ns)	Bandwidth(GB/s)
HPE	0: Local	97	101
	1: 1 Hop	226	16
	2: 2 Hop	260	16
	3: Remote	380	12
Power8	0: Local	117	193
	1: 1 Hop	142	28
	2: n/a		
	3: Remote	260	43
Power9	0: Local	118	148
	1: 1 Hop	214	39
	2: n/a		
	3: Remote	361	90

and 1536 logical cores (hardware threads), of which 752 and 1504 are available to applications. Both systems provide 16 TB of memory.

The IBM Power CPUs use a RISC-based instruction set architecture (ISA) unlike Intel CPUs, though both types of CPUs share many features such as vector instructions, support of hardware transactional memory, and simultaneous multithreading (SMT). Notably, IBM Power CPUs realise configurable levels for SMT exposing 1 to 8 logical cores (hardware threads) per physical CPU core. In our experiments, we use SMT-8 if not noted otherwise. Therefore, the 12 physical cores in either Power processor provide up to 96 logical cores (hardware threads). We remark that both IBM Power platforms reserve one physical core on some CPUs (2 on our Power8 and 4 on our Power9) to manage logical hardware partitions (LPARs), e.g. governing storage and other periphery. All other cores and memory resources are available for our evaluation without restrictions and any resource sharing.

In the memory hierarchy, caches and the NUMA hierarchy differ in important aspects between the two Power platforms as well as the Intel-based platform. The cache capacities in Table 2b reveal the large L3 caches per physical core for Power9 and Power8. The Power9 processor contains the largest L3 cache per physical core. Additionally, higher overall cache performance is claimed on Power9 due to increased associativity (20-way in Power9 vs. 8-way on Power8) [49]. Notably, the IBM systems have a L4 cache on their custom DRAM DIMMs. However, this is a memory buffer outside of the processor rather than a CPU cache, thus unlike CPU caches this L4 cache does not hide NUMA effects.

Like the Intel-based platform, the IBM Power systems follow a NUMA architecture to scale to more than 1000 logical cores. In Table 2c, we summarise the respective latency and bandwidth of memory accesses alongside the Intel-based platform. As shown in Fig. 1c, our 16-socket Power9 system features a NUMA topology with one hop to sockets in the same chassis and two hops to sockets in its other three chas-

sis, whereas our smaller 8-socket Power8 system can afford a fully connected NUMA topology directly connecting every socket between its two chassis. Additionally, both systems have faster interconnects within the chassis than between the chassis, therefore still establishing three NUMA level on both Power systems.

Notably, a larger Power8 system with 16 sockets would have a similar topology to our Power9 and inversely smaller versions of the HPE and Power9 systems would similarly benefit from stronger connections.

2.3 Benchmarking environment

In this paper, we evaluate the CC schemes mentioned before on our multi-socket hardware with the TPC-C benchmark [56] as implemented in the latest version of DBx1000 [72]. This version of DBx1000 includes the extended set of CC schemes as mentioned before and bug fixes, beyond the version used in the original paper [71]. Additionally, we rely on the provided embedded instrumentation to measure the time spent in the system.

For running the benchmarks, we use the given default configuration of DBx1000. This configuration defines the TPC-C workload as equal mix of *New-Order* and *Payment* transactions covering 88% of TPC-C with standard remote warehouse probabilities (1% and 15%). This configuration partitions the TPC-C database by warehouse (WH) ID for all CC schemes. Based on this configuration, we specify 4 warehouses for the high conflict TPC-C workload and 1024 or 1568 warehouses for the low conflict workload, as in our initial DaMoN paper [3]. Similar to the original evaluation, each benchmark runs until the first transaction executor has committed 100 K transactions, and we measure the throughput as the number of transactions committed by all transaction executors in that time. We observed this method to provide reliable measurements despite NUMA effects in our large system or other effects influencing the execution of the benchmark.

An interesting first observation was that DBx1000's TPC-C did not implement insert statements, presumably due to the mentioned limitations of the simulator, e.g. memory capacity and no OS. In part one of the paper, we thus first start with the very same setup, but later we enable insert statements in the evaluation after taking a first look at the CC schemes. As minor extension, we added a locality-aware thread placement strategy to DBx1000 for all experiments in this paper. It exclusively pins DBMS threads to a specific core. For scaling the DBMS threads in our experiments, we use the minimal number of sockets to accommodate the desired resources, e.g. 2 sockets for 112 threads, otherwise OS and NUMA effects would dominate the overall results. Note that as consequence of this thread placement strategy, *cores* and *threads* equally refer to a single execution stream (i.e. a worker) of the DBMS.

For the broader evaluation in part two, we use a setup similar to the optimised DBx1000 from part one as discussed before. Notably, we aim to match the thread placement on the additional hardware platforms with the placement we used on HPE. Moreover, throughout part two, TPC-C is configured with the full TPC-C schema and transactions always execute their insert statements. Finally, for the last experiments in part two, we extend DBx1000 to support the remaining TPC-C transactions for the full TPC-C benchmark. For all other experiments in part two, we use the more narrow mix of *New-Order* and *Payment* transactions only, as was the case for the original simulation.

3 Part one: simulation vs. real hardware

3.1 A first look: simulation vs. reality

We now report the results of running the DBx1000 prototype directly on the multi-socket Intel-based hardware (HPE platform) as opposed to a simulation.

3.1.1 The plain results

Figure 2 displays the throughput of TPC-C transactions for 4 warehouses and 1024 warehouses, i.e. high and low conflict OLTP workloads. On the left of each figure are the original simulation results [71] and on the right are our results on real multi-socket hardware (Intel-based HPE). We first compare

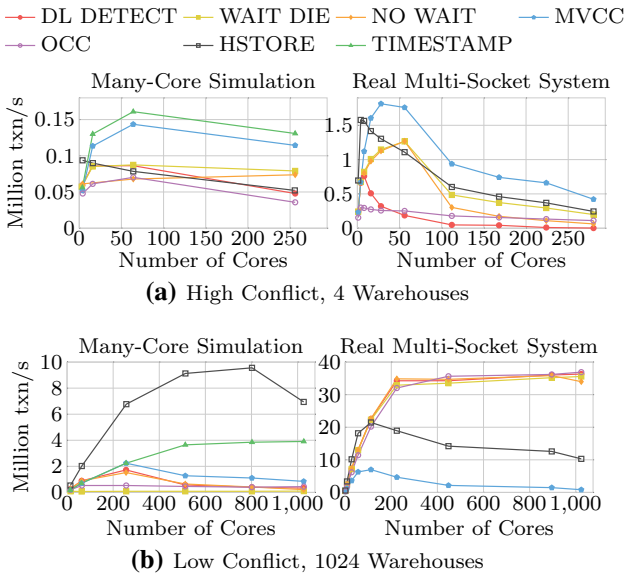


Fig. 2 Throughput of TPC-C in original simulation [71] and on real multi-socket hardware (Intel, HPE)

the overall throughput. Then, we break down where time is spent in the DBMS to better understand our observations.

We first look at the results for 4 warehouses as shown in Fig. 2a. Overall, it is obvious that the absolute throughput differs due to the characteristics of the CPUs in the simulation and our hardware, e.g. low-power 1 GHz cores versus high-power 2.5 GHz cores. This is expected, and therefore, only the relative performance of the CC schemes matters. In the following, we discuss similarities and significant differences.

First, comparing the simulation and the real hardware (Intel-based HPE platform) in Fig. 2a, we see that the CC schemes HSTORE, MVCC, and NO WAIT show similar trends. That is, these CC schemes have a similar thrashing point in the simulation and the real hardware, i.e. HSTORE at 4–8 cores and MVCC as well as NO WAIT at 56 to 64 cores. After the respective thrashing point, these CC schemes degrade steeper on the multi-socket hardware, which can be linked to the additional NUMA effect of the multi-socket hardware appearing beyond 56 cores. For the other CC schemes, the results for the simulation and real hardware differ more widely, especially the diverging behaviour of the pessimistic CC schemes sticks out. Considering these pessimistic CC schemes, DL DETECT behaves broadly different already degrading at 8 cores rather than 64 cores and WAIT DIE performs surprisingly close to NO WAIT. In Sect. 3.1.2, we analyse the time breakdown of this experiment to explain these results. It reveals characteristic behaviour of the individual CC schemes, despite the diverging throughput in the simulation and the multi-socket hardware.

Next, we look at the low conflict TPC-C workload (1024 warehouses) in Fig. 2b. The results here present fewer similarities of the many-core simulation and the multi-socket hardware (Intel-based HPE), i.e. only the slope of MVCC is similar. Additionally, DL DETECT and NO WAIT stagnate at high core counts (>224) in the simulation and on the multi-socket hardware. In contrast, HSTORE performs worse on the multi-socket hardware than in the simulation. It is slower than the pessimistic CC schemes and OCC from >112 cores. Also, OCC and WAIT DIE achieve higher throughput on the multi-socket hardware, now similar to DL DETECT and NO WAIT. Moreover, MVCC is significantly slower than OCC and the pessimistic CC schemes, due to high overheads of this scheme as we discuss next.

Insight The initial comparison of concurrency control schemes on 1000 cores presents only minor similarities between the simulation and our multi-socket hardware with surprising differences in the behaviour of the CC schemes mandating further analysis.

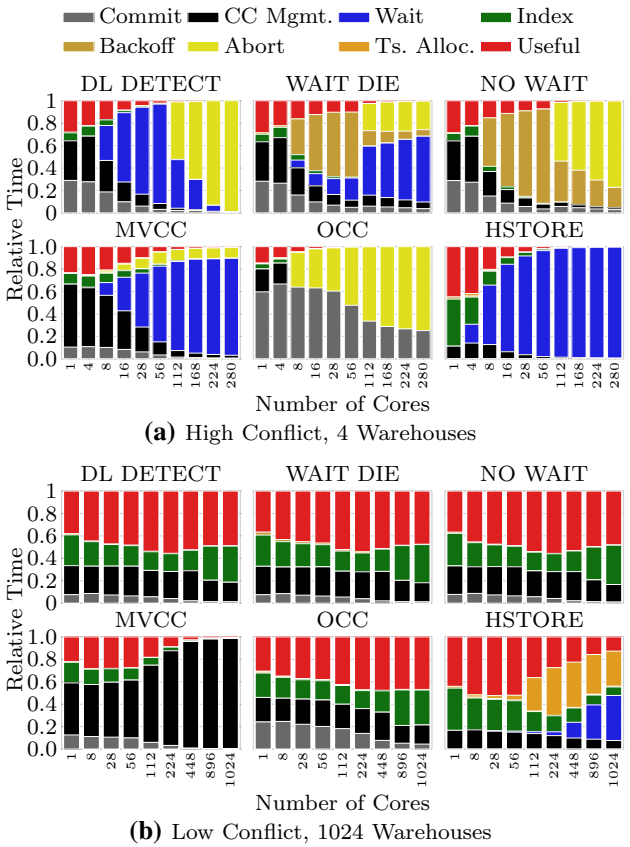
3.1.2 First time breakdown on intel-based hardware

For deeper understanding of the observed behaviour of the CC schemes, we now break down where time is spent in processing the TPC-C transactions on the multi-socket hardware. Therefore, we apply the breakdown of [71] categorising time as outlined in Table 3. For each CC scheme, Fig. 3 breaks down the time spent relative to the total execution time of the TPC-C benchmark with a bar for each core count.

The time breakdown of 4 warehouses in Figure 3a neatly shows the expected effect of conflicting transactions and aborts for increasing core counts under high conflict work-

Table 3 Time breakdown categories

Useful	Time usefully executing application logic and operations on tuples.
Abort	Time rolling back and time of wasted useful work due to abort.
Backoff	Time waiting as backoff after abort (and requesting next transaction to execute).
Ts. Alloc.	Time allocating timestamps.
Index	Time operating on hash index of tables including latching.
Wait	Time waiting on locks for concurrency control.
Commit	Time committing transaction and cleaning up.
CC Mgmt.	Time managing concurrency control other than prior categories, e.g. constructing read set.



(a) High Conflict, 4 Warehouses

(b) Low Conflict, 1024 Warehouses

Fig. 3 Breakdown of relative time spent for TPC-C transactions on multi-socket hardware (Intel, HPE)

load. That is, most CC schemes result in high proportions of *wait*, *abort*, and *backoff* as soon as the number of cores exceeds the number of warehouses (>4 cores), yielding nearly no *useful work* at higher core counts. Only the *wait* time of HSTORE grows at 4 cores concurrently executing transactions, such that HSTORE appears more sensitive to conflicts.

Remarkably, textbook behaviour of the specific schemes becomes visible in the breakdown: Starting with DL DETECT, its *wait* time increases with the number of concurrent transactions as expected, following the increasing potential of conflicts between concurrent transactions. Different from DL DETECT, WAIT DIE spends more time *backing off* and *aborting* due to its characteristic aborts after a short wait time (small wait proportion). Instead, NO WAIT solely *backs off* without waiting, spending even more time on *aborted* transactions. The optimistic MVCC waits on locks during validation, such that its breakdown shows similar *wait* times like DL DETECT. Finally, for OCC we can see that the high *abort* portion reflects its sensitivity to conflicts while the high *commit* portion stems from high costs for cleaning up temporary versions.

Having observed this "expected" behaviour of the CC schemes under high conflict, we now analyse the unexpected behaviour under low conflict as shown in Fig. 3b. Against the expectation, most CC schemes spend considerable amount of time to manage concurrency (black and grey area) such as lock acquisition (except HSTORE which we discuss later). For these schemes, this results in at most 50% of useful work (red area). Staggeringly, MVCC, which actually should perform well under low conflicting workloads, spends almost no time with *useful work* despite the low conflict in the workload, i.e. <10% *useful work* from 224 cores. In fact, the low conflict is visible in the overall little time spent *waiting* or *aborting*. Consequently, the slowdown compared to pessimistic CC schemes does not stem from wasted work but from internal overhead in execution of this CC scheme under high core counts.

In contrast, we observe for HSTORE an increasing impact of *timestamp allocation* and *waiting time*. While *timestamp allocation* is used by the other schemes as well, the relative overhead for HSTORE is the highest due to its cheaper lock acquisition. In fact, the authors of [71] did analyse different timestamp allocation methods in their paper but chose *atomic increment* as a sufficiently well performing method that is a generally applicable option when there is no specialised hardware available. However, as we can see this choice is not optimal for multi-socket hardware. Moreover, we attribute the increasing *waiting time* of HSTORE to its coarse-grained partition locking to sequentially execute transactions on each partition. This partition-level locking causes a higher overhead if more cores are used since this leads to more conflicts between transactions as shown in prior work [34,41].

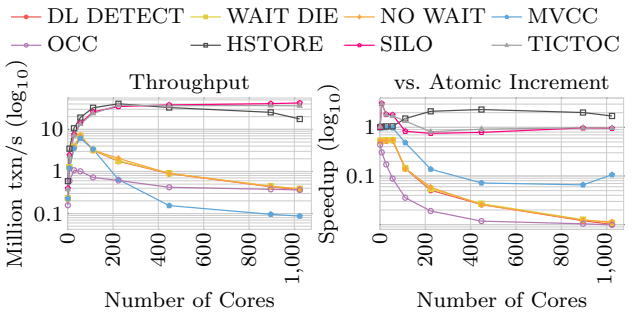


Fig. 4 Throughput of low conflict TPC-C for timestamp allocation with hardware clock

Insight The analysed CC schemes behave differently on the real multi-socket hardware than in the simulation of [71]. For the high conflict workload (4 warehouses), the behaviour on real hardware and the simulation appears more similar, for which the time breakdown confirms the expected characteristics for each CC scheme. However, low conflict workload causes an unexpectedly high CC management overhead in most CC schemes and transactions execute only a limited amount of useful work, except for HSTORE where waiting and timestamp allocation dominate.

3.2 A second look: hidden secrets

In this section, we now take a "second look" at the factors leading to the surprising behaviour of the CC schemes observed in our initial analysis and discover equally surprising insights.

3.2.1 Hardware assistance: the good?

In a first step, we analyse the benefit of hardware-assisted timestamp allocation over using atomic counters for the real multi-socket hardware. As explained earlier, the *atomic increment* is generally applicable but may cause contention, which efficient and specialised hardware may prevent if available. Our hardware provides a synchronised hardware clock indeed offering a new option for efficient timestamp allocation, as projected by [71]. Specifically, our Intel processors provide efficient access to a hardware clock on each core [29] (*rdtsc* instruction), which ticks at the same rate on all cores in the entire system (*invariant tsc* feature) and is synchronised across all cores and hardware chassis by the platform firmware, verified by the OS [43,59–61].

In the following experiment, we analyse the benefit of this hardware assistance for timestamp allocation. Figure 4 shows the throughput of the CC schemes for 1024 warehouses with timestamp allocation based on the hardware clock.

On one hand, HSTORE greatly benefits from the hardware clock (as expected) achieving peak throughput of

∼40 M txn/s with an overall speedup over atomic increment of up to 3x. We now also include SILO and TICTOC in our results which perform like HSTORE except for high core counts as we discuss in the time breakdown analysis below. On the other hand, the remaining CC schemes (DL DETECT, WAIT DIE, NO WAIT, MVCC, and OCC) degrade drastically when using the hardware clock instead of atomic counters. That is, the pessimistic CC schemes DL DETECT, WAIT DIE, and NO WAIT perform ∼50% slower within a socket (0.51 - 0.55x speedup for ≤56 cores), after which they degrade to 0.01x speedup at 1024 cores (0.37 - 0.39 M txn/s). Likewise, MVCC is stable (∼1x speedup) up to 56 cores and its speedup drops to 0.1x when exceeding the single socket. Finally, OCC does not benefit from the hardware clock at all (0.44 - 0.01x speedup).

Overall, timestamp allocation based on the hardware clock drastically changes the perspective on the performance of the CC schemes. Now, HSTORE performs best, meeting the initial observations of [71] (joined by SILO and TICTOC), whereas the pessimistic schemes, OCC, and MVCC degrade severely.

For better understanding of these diverse effects of the hardware clock, we again look at the time breakdown shown in Fig. 5a (top row). As expected, HSTORE now spends little time for timestamp allocation (like SILO and TICTOC). Otherwise, HSTORE spends time similarly as with atomic increment, especially with a similar increase in the *waiting time*. Importantly, we do not observe any bias introduced by the hardware clock, since HSTORE (as well as the other CC schemes) spends no significant time aborting and our detailed logs show no outliers for the number of aborted transactions per transaction executor. Consequently, the hardware clock is reliable for timestamp allocation.

An interesting observation is the significant change in the time breakdown of the other CC schemes. For example, DL DETECT, WAIT DIE, and NO WAIT show at least 2x the time spent for *CC Mgmt.* and *committing/cleaning up* (black & grey) with a sudden increase after 56 cores. OCC's increase in time spent in these categories is even more drastic with less than 20% of *useful work* at any core count. Only MVCC changes insignificantly, as *useful* time spent was low already.

Profiling these CC schemes reveals physical contention, that previously was on the atomic counter, now results in thrashing of latches. Previously, the physical contention on the atomic counter has throttled transaction execution including latching, e.g. for lock acquisition. Now, that the hardware clock has removed the physical contention from timestamp allocation, transactions access latches more frequently, indeed reaching their thrashing point despite a latch per row and low conflicts in the workload with 1024 warehouses. Notably, our profiling reveals further details of the individual CC schemes: The *pthread_mutex* employed in DL DETECT, WAIT DIE, NO WAIT, and OCC sharply degrades due to NUMA sensitivity of hardware transactional memory [9] used for lock elision and its fallback to robust but costly queuing synchronisation [18] as well as costly interaction with the scheduler of the OS.[2] In contrast, MVCC uses an embedded flag as spin latch which is not as sensitive to NUMA but also not robust [12]. Hence, this type of latch shows a slower but also continuous degrading of performance.

Insight Hardware-assisted timestamp allocation via specialised clocks alleviates contention and leads to better scalability for HSTORE (as well as SILO and TICTOC). However, while hardware-assisted clocks also lift the overhead in the other schemes, it does not necessarily improve their overall performance as contention moves and puts pressure on other components (e.g. latches), even leading to performance degradation.

3.2.2 Data size: the bad?

In the context of this surprisingly high overhead, our second look at the paper [71] brings the following statement to our attention: "Due to memory constraints [...], we reduced the size of [the] database" [71]. Consequently, we are wondering if the staggering overhead is potentially caused by the absence of useful work to execute rather than the abundance of overhead in the CC schemes, due to the reduced data size imposed by limited memory capacity of the simulator in [71].

We revert the benchmark to the full TPC-C database in the following experiment and report on the surprising effect of the larger data volume. In detail, to return to the officially specified database, we increase (1) the cardinality of the item relation from 10 K to 100 K, (2) the factor of customers per warehouse from 20 K to 30 K determining the cardinalities of the customer, order, order-line, and history relations, and (3) we include all attributes rather than only those accessed.

Figure 6 shows the throughput for the full schema with 1024 warehouses and speedup in comparison with the small schema based on the previous experiment (cf. Fig. 4). We measure quite diverse throughput of the CC schemes. Yet, the speedup indicates that two major effects of the increased data volume appear in the same clusters as in the previous experiment but with inverse outcome. The first cluster of HSTORE, SILO, and TICTOC is slower with the full schema, i.e. 0.2–0.6x, 0.3–0.5x, and 0.2–0.5x, respectively. The second cluster, consisting of the previously "slower" CC schemes, improves inversely to the previously described thrashing points. That is, DL DETECT, WAIT DIE, and NO WAIT have a speedup of 0.7x until 56 cores, after which they benefit from the full schema with speedups of 2.4–9.1x, 2.5–8.3x, and 2.0–9.3x, respectively. MVCC has a speedup of 0.5–0.6x until 56 cores, breaks even (1x) at 112 cores

[2] pthread_mutex is specific to *libc* and the OS.

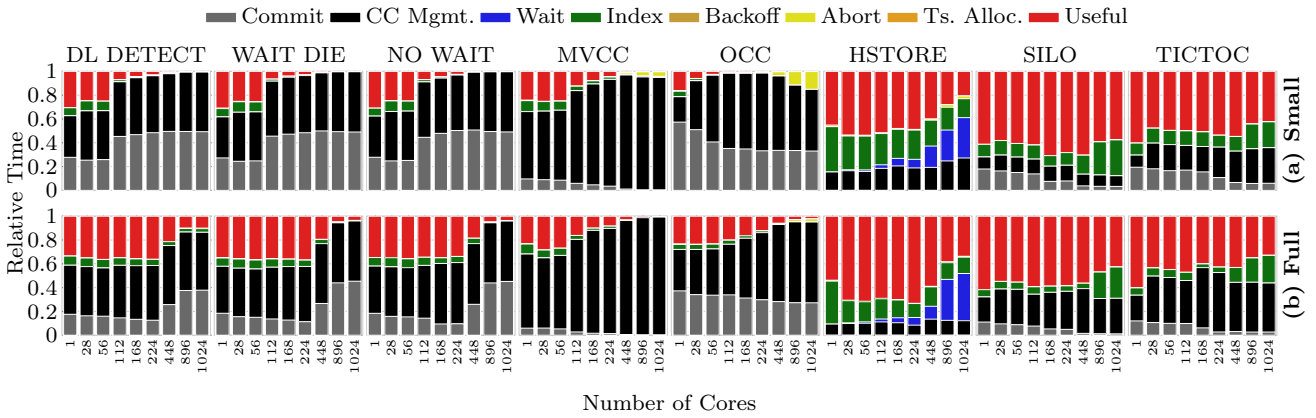


Fig. 5 Breakdown of relative time spent processing TPC-C transactions on *small* and *full* schema with 1024 warehouses using timestamp allocation via hardware clock

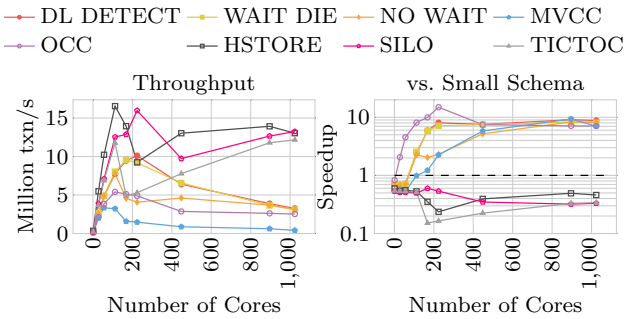


Fig. 6 Throughput of low conflict TPC-C for small schema size like in the simulation versus full schema size both executed on multi-socket hardware (Intel, HPE)

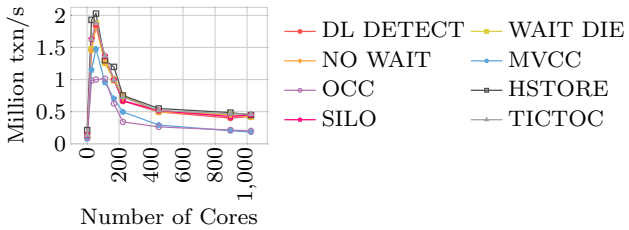


Fig. 7 Throughput of TPC-C including inserts with full schema size on multi-socket hardware (Intel, HPE)

and then, improves with a speedup of 1.2–9.3*x*. OCC has a speedup of 0.8 at 1 core and broadly improves with the full schema with 2.1–14.9*x* speedup.

The time breakdown in Fig. 5b (lower row) details the causes. As for the CC schemes in the first cluster, HSTORE has increased *useful* work, while for SILO and TICTOC *CC Mgmt.* increases. Both indicate increased cost of data movement, as HSTORE directly accesses tuples and the other two create temporary copies in the CC manager. The second cluster shows an increase in *useful* work, presenting less staggering overhead of CC management at low core counts. Importantly, the sudden increase in *commit* for DL DETECT, WAIT DIE, and NO WAIT is delayed, indicating that latches thrash only from 448 cores (while previously already from 112 cores). For OCC, the time spent on *commit* also decreases with the larger data volume, but the increase in *CC Mgmt.* due to larger temporary copies still diminishes *useful work*. Only for MVCC, the time breakdown does not change significantly.

Insight The effect of larger data volumes in the full schema changes the perspective on the CC schemes again. We attribute our observations to the effects, that heavier data movement slows down data-centric operations (e.g. tuple accesses or copies), which in turn alleviates pressure on latches preventing thrashing.

3.2.3 Inserts: facing reality!

Since the simulator of [71] had limited memory capacity and excluded the simulation of important OS features such as memory management, the TPC-C implementation of DBx1000 did not include insert statements and for comparability we initially excluded these as well. For the last experiment in this section, we now complete the picture of concurrency control on real hardware (Intel).

Accordingly, Fig. 7 shows the throughput of TPC-C transactions including inserts (as well as all before-mentioned changes) for 1024 warehouses. The inserts drastically reduce throughput of all CC schemes with heavy degradation at the socket boundary (56 cores). Even more interesting, all CC schemes perform similarly with inserts in the transactions. Indeed, profiling indicates execution of insert statements are the hotspot of the TPC-C transactions now, but the causes are orthogonal to concurrency control. The two major hotspots are (1) catalogue lookups to locate tuple fields and (2) memory allocation for new tuples.

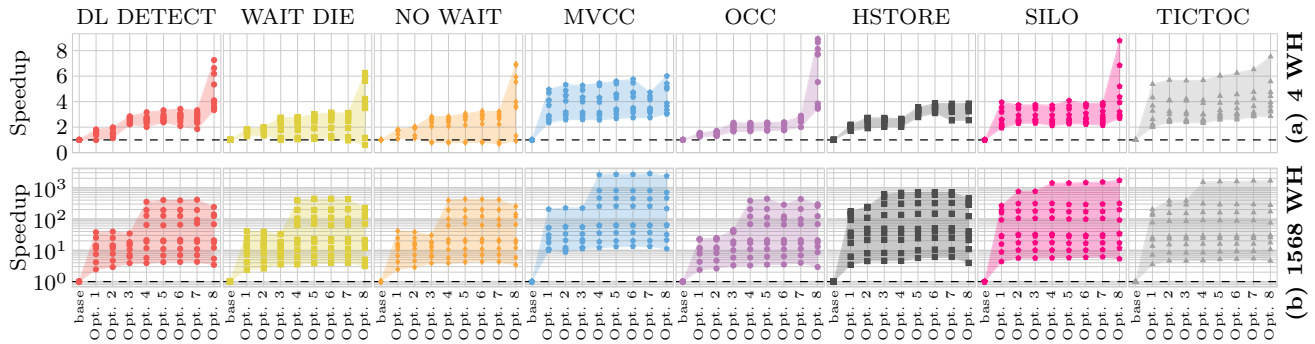


Fig. 8 Summary of speedup of the CC schemes provided our optimisations Opt. 1 - 8 described in Sect. 3.3.1 for (a) the high and (b) the low conflict workloads (i.e. 4 and 1568 warehouses). Optimisations are applied one after the other, and speedup is reported as the factor of throughput increase over the base implementation

Profiling details show that catalogue lookups cause frequent accesses to L1 and L3 caches. For tuple allocation, profiling details indicate significant time spent in the memory allocator and for OS memory management including page faults. These hotspots are amplified by NUMA in our multi-socket system, since the catalogue is centrally allocated and memory management in Linux is also contention- and NUMA-sensitive [11]. Hence, such impact on performance only becomes visible in its full extent on large systems like ours.

Insight Inserts significantly affect the performance in this benchmark, though due to hotspots orthogonal to the CC schemes, most notably cache misses of the catalogue and memory allocation.

3.3 Effect of state-of-the-art-optimisations

Finally, we take a last step to provide a clear view on the characteristics of concurrency control on large multi-socket hardware. First, we elaborate on our optimisations to reach this clear view and provide an overview of their individual speedups. Then, we repeat our assessment of the CC schemes using all optimisations.

Notably, in the following experiments, we use the full TPC-C schema as well as inserts in the transactions, and we exercise the whole 1568 cores for the low conflict workload. We maintain the one-to-one relation of cores to warehouses for the low conflict workload, as the TPC-C workload induces significant conflict when concurrent transactions exceed the number of warehouses.

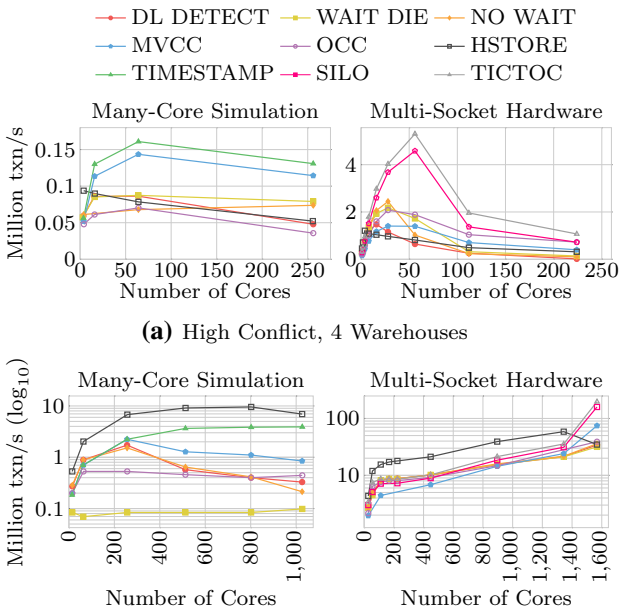
3.3.1 Overview of optimisations

To clear the view, we remove previously identified obstacles and optimise the overall system based on state-of-the-art in-memory DBMS for large multi-socket hardware: (Opt. 1) We introduce a thread-local memory allocator that pre-allocates memory, as in today's commercial in-memory databases [16]. Importantly, it aligns allocations to cache line boundaries, otherwise false sharing obliterates performance. (Opt. 2) We add a flat perfect hash index [34], e.g. reducing pointer chasing and cache misses. (Opt. 3) We address latch thrashing with a queuing latch [12,34,54] and exponential backoff [26]. (Opt. 4) We replicate read-only relations to each socket, utilising faster local memory instead of slow remote memory [34]. (Opt. 5) We reorder and prefetch tuple and index accesses to optimise data movement. (Opt. 6) We lift expensive query interpretation (e.g. catalogue lookups) to efficient query compilation as in state-of-the-art in-memory DBMS [16,33,55,57]. (Opt. 7) We update the deadlock prevention mechanisms to state-of-the-art [26]. (Opt. 8) We eliminate CC overhead for read-only relations.

In Fig. 8, we report the speedup provided by each optimisation when consecutively adding the optimisations, as the factor of throughput increase over the unoptimized *base* implementation. Note that, we discuss the detailed throughput with all optimisations in place in the next section and the detailed throughput of the individual optimisations is available in [5].

For the high conflict workload, Fig. 8a shows that the thread-local memory allocator (Opt. 1) and the eliminated CC overhead for read-only relations (Opt. 8) provide significant speedup for all CC schemes with each up to 5.38x and 4.33x. Additionally, the optimised latching (Opt. 3) indeed notably benefits the CC schemes involving heavy latching (pessimistic CC schemes and OCC) with further speedup of up to 2.12x.

For the low conflict workload, Fig. 8b shows an even greater speedup for the thread-local memory allocator (Opt. 1), by up to 268x. Additionally, in this low conflict workload the NUMA-aware replication (Opt. 4) proves beneficial with up to 227x speedup, as actual work including record accesses dominates the low conflict workload (rather than concurrency control). Further, there are notable speedups of



(a) High Conflict, 4 Warehouses

(b) Low Conflict, 1024 Warehouses for Simulation and 1568 for Multi-Socket Hardware

Fig. 9 Throughput of TPC-C in original many-core simulation [71] without full schema & inserts and our optimised implementation with full schema & inserts on multi-socket hardware (Intel, HPE)

the other optimisations for distinct CC schemes, e.g. the optimised index (Opt. 2) notably benefits HSTORE, SILO, and TICTOC with up to 2.83*x*, whereas the optimised latching (Opt. 3) again benefits the pessimistic CC schemes, OCC, and now also HSTORE (up to 2.49*x*). Finally, the updated deadlock prevention (Opt. 7) significantly benefits OCC with up to 1.71x. The other optimisations show less significant speedups (<1.5*x*).

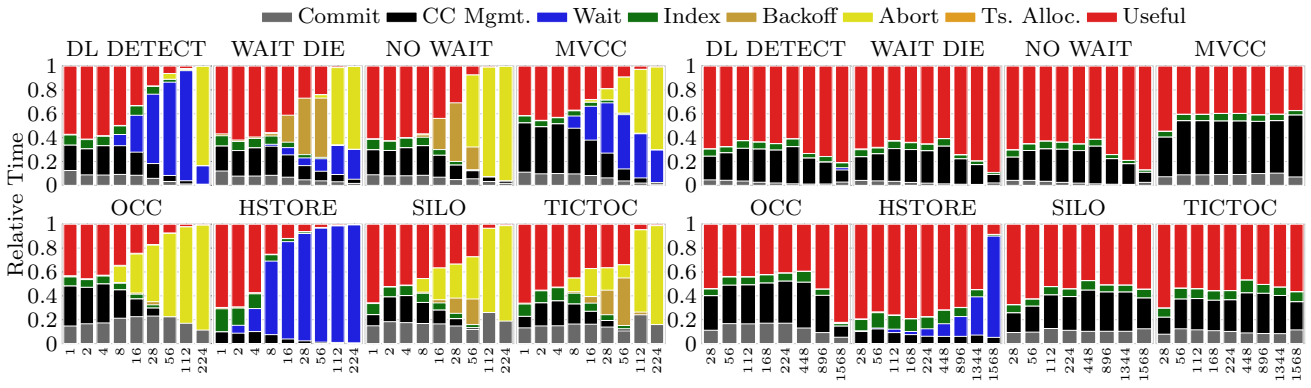
Notably, the speedup of the optimisations varies in detail, across the CC schemes, workloads, and number of cores. There are many factors influencing the specific speedup. Their detailed study is beyond the scope of our evaluation of concurrency control.

3.3.2 Results after optimisations

With the above optimisations in place, we now repeat the detailed assessment of the CC schemes under high and low conflict OLTP workload (as initially in Sect. 3.1). Accordingly, Fig. 9 presents the throughput of the fully optimised DBx1000 for the high conflict and low conflict TPC-C workloads. In addition, Fig. 10 again details the performance of the CC schemes on the multi-socket hardware with time breakdowns.

Starting with throughput of the high conflict workload in Fig. 9a (top row), we again observe similar results as reported in our first assessment. The many-core simulation and the multi-socket hardware results show different but reasonable behaviour due to the respective hardware characteristics. The only difference is that now our optimisations further offset throughput on the multi-socket hardware. Additionally, we now include the advanced CC schemes SILO and TICTOC whose peak throughput remarkably outperform the originally covered CC schemes with 4.6 and 5.3 M txn/s, respectively. Yet, those two CC schemes similarly degrade at high core counts converging to the performance of the other CC schemes from 56 cores (>1 socket).

For the other CC schemes, there are minor similarities of the individual throughput curves of the CC schemes between the many-core simulation and the multi-socket hardware. Focusing on the relative performance of the CC schemes other than SILO and TICTOC reveals significant improvement of OCC and decrease in MVCC. The pessimistic schemes converge at high core counts, only degrading at different points and rates. Finally, HSTORE still only performs



(a) High Conflict, 4 Warehouses

(b) Low Conflict, 1568 Warehouses

Fig. 10 Breakdown of relative time spent processing TPC-C transactions with optimised DBx1000 using full schema and inserts on multi-socket hardware (Intel)

well for small core counts (≤4) and remains slow beyond. Moreover, considering the time breakdown for the high conflict TPC-C workload in Fig. 10a, we again observe textbook behaviour as in the early time breakdown in Sect. 3.1.2 with fractions of *wait, backoff*, and *abort* characteristic for the individual CC schemes, though the amount of *useful* generally improves and *commit* as well as *CC Mgmt.* decrease through our optimisations.

Next, we analyse the low conflict workload using our optimised implementation. Figure 9b reveals that under this workload all CC schemes broadly provide scalable performance with fewer differences as the schemes show in the many-core simulation. That is, up to two sockets the throughput of all CC schemes steeply grows. Then, the throughput continues to grow linearly up to 1344 cores at a lower growth rate. At the full scale of 1568 cores, the behaviour of the CC schemes differs. TICTOC, SILO, and MVCC make a steep jump reaching 197 M txn/s, 159 M txn/s, and 75 M txn/s, respectively. Also, the growth rate of the pessimistic locking schemes increases but not as much, yielding 34 M txn/s for DL DETECT, 32 txn/s for WAIT DIE, and 36 M txn/s for NO WAIT. OCC stays linear achieving 39 M txn/s. In contrast, HSTORE degrades from 59 M txn/s at 1344 cores to 35 M txn/s at 1568 cores.

Now with this clear view, we can make out different characteristics of the CC schemes on the large multi-socket hardware, visible in their throughput and time breakdown (Fig. 10b). Under high conflict, the schemes SILO and TICTOC clearly excel, although they neither scale to high core counts (similar to the other schemes). Under low conflict, HSTORE performs the best until the number of concurrent transactions (cores) equals the number of partitions (warehouses). Beyond this point it degrades, due to its coarse partition locking, similarly observed in the simulation. HSTORE's sensitivity to conflicts becomes obvious in the steep increase in *wait* time in the time breakdown.

Under low conflict, TICTOC follows as second fastest with SILO close by. Both provide significantly lower throughput than HSTORE until the tipping point at 1344 cores from which they outperform HSTORE by a large margin due to efficient fine-grained coordination, as indicated by their stable amount of *Commit* and *CC Mgmt.* For the other CC schemes, the view is diverse as their relation changes with the NUMA distance between the participating cores. After exceeding 8 sockets (448 cores/2 chassis), the pessimistic schemes fall behind the advanced optimistic CC schemes (TICTOC & SILO) and eventually also behind OCC and MVCC. This degrading is unrelated to conflicts (no *wait* time) but correlates with increasing NUMA distances. Consequently, for the low conflict OLTP workload, it appears that pessimistic locking is beneficial when access latencies (NUMA effects) are low. The temporary copies of optimistic CC can hide these latencies, but at the cost of additional data movement, slowing down throughput at close NUMA distance. To this end, HSTORE and TICTOC implement these two approaches as well, but they are more efficient, e.g. as HSTORE locks less frequently. Notably, there is no difference among the pessimistic CC schemes with different mechanisms against deadlocks, as the low conflict has few deadlocks.

Insight After spending considerable engineering effort bringing state-of-the-art in-memory optimisations to DBx1000, we shed new light on concurrency control on 1000 cores. First, we unveil remarkable peak throughput of the newer CC schemes, TICTOC and SILO, on high conflict workload, while also presenting textbook behaviour of all CC schemes in the time breakdown. Second, we brighten the grim forecast of concurrency control on 1000 cores for low conflict workload from the simulation of [71]. In fact, under low conflict all CC schemes scale nearly linearly to 1568 cores reaching 200 million TPC-C transactions per second.

3.4 Summary of part one

In this part, we analysed in-memory DBMS on an Intel-based platform with 1568 cores, revisiting the results of the simulation in [71], using their original prototype DBMS DBx1000. This led to surprising findings:

(1) A first attempt of running their prototype on today's multi-socket hardware presented broadly different behaviour of the CC schemes. To our surprise, the low conflict TPC-C workload with at most one warehouse per core (and transaction executor) revealed most concurrency control schemes not only stopped scaling beyond 200 cores but also were very inefficient spending not even half of their time on useful work.
(2) Based on these results, we decided to take a second deeper look into the underlying causes and made several discoveries. First, the default timestamp allocation via atomic increment was a major bottleneck on the multi-socket hardware. Second, the default benchmark settings of DBx1000 used a TPC-C database significantly reduced in size and disregarded inserts in the transactions. Changing these default setting shifted the picture of our initial assessment completely: while replacing the atomic counter with a hardware clock removed the timestamp creation bottleneck, enabling the original database size and insert statements, however, led to an even darker picture than in our first look. In this second look, we saw that all CC schemes completely collapsed when scaling to more than 200 cores, despite absent conflicts in the workload.
(3) Finally, we spent significant engineering efforts on our optimised DBx1000 [6] across all components from memory management over transaction scheduling to

locking. This cleared the dark skies we faced before and allowed most CC schemes to scale very well, providing up to 200 million txn/s on 1568 cores. Even more surprisingly, now, the CC schemes behave very similar with no clear winner. Having cleared the view on concurrency control with this evaluation on real hardware, an interesting question is now how these findings generalise across different scale-up hardware platforms and more demanding workloads.

4 Part two: broadening the evaluation

In this second part, we broaden the evaluation of in-memory OLTP DBMS on large hardware. Previously, in the first part, we evaluated how the insights of in-memory DBMS running on a simulated many-core hardware transfer to today's hardware. Indeed, we observed significantly different behaviour of the in-memory DBMS DBx1000 on real hardware compared to the original simulation [71]. For the second part, we now widen the evaluation in the two dimensions hardware and workload, as discussed in Sect. 1. First, we study the CC schemes on a broader set of hardware platforms, before we then look at the full TPC-C transaction mix.

4.1 Intel-based vs. IBM power 8/9 platforms

We begin with an overview how the different approaches to "1000 cores" of today's hardware affect concurrency control. Initially, we focus on identifying diverging behaviour of CC schemes on the different hardware platforms, performing scalability experiments. Later sections cover detailed root cause analyses. The following scalability experiments determine how the CC schemes respond to increasing number of cores provided by the three different hardware platforms (HPE, Power9, and Power8), when the CC schemes pressure different aspects of the hardware depending on the scale (number of cores). For example, compute resources, caches, and interconnects between the processors are utilised differently depending on the hardware scale (number of cores). As before, we separately evaluate high and low conflict workload, due to their significant effect on the CC schemes. For example, high conflict generally requires more coordination (e.g. latching), while low conflict allows for high concurrency, influencing the behaviour of the CC schemes on the different platforms.

4.1.1 Scaling on different real hardware—high conflict

Figure 11 presents the performance of the CC schemes for the high conflict workload on HPE, Power9, and Power8. Overall, for the throughput in Fig. 11a, we observe vaguely similar scaling behaviour on Power9 and Power8 as previously on HPE, i.e. the CC schemes briefly scale well but eventually thrash. This thrashing is caused by the high conflict in the workload. As indicated by according abort rates in Fig. 11b, the CC schemes respond to these conflicts similarly on all three hardware platforms. Notably, not only the general behaviour is similar on the three platforms, but also the actual throughput of the CC schemes is of the same magnitude as opposed to the simulation, allowing for comparison of absolute performance.

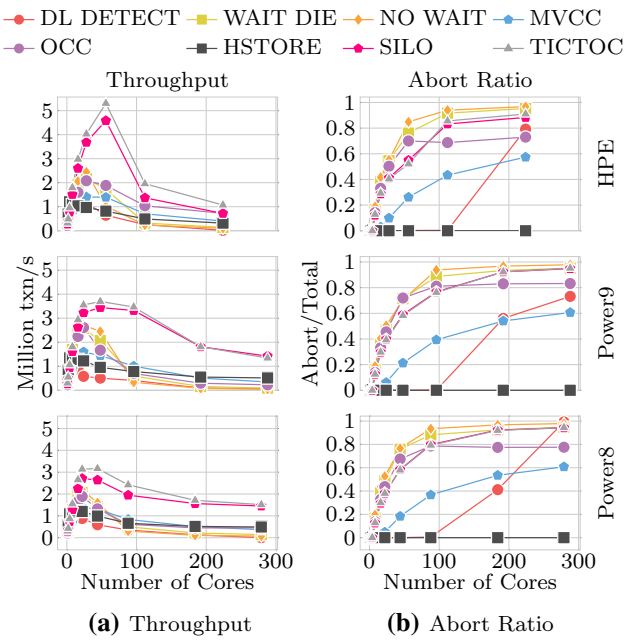


Fig. 11 Performance for TPC-C under high conflict on HPE, Power9, and Power8

Figure 12 details the scaling behaviour of the individual CC schemes on the three hardware platforms side by side (i.e. 1. HPE, 2. Power9, and 3. Power8). As discussed next, their diverse behaviours indicate no clear benefit of either hardware platform, but rather highlight the benefit of individual hardware properties taking effect at specific core counts.

Starting with the pessimistic locking scheme DL DETECT, we find its peak performance on HPE and at only 16 cores (1.5 M txn/s, 5.3x). On Power9, DL DETECT sharply degrades already at 16 cores, falling behind the performance on HPE and Power8. Beyond 16 cores, DL DETECT degrades on all three hardware platforms. Instead, the other two pessimistic locking schemes WAIT DIE and NO WAIT achieve their peak performance at 24 cores on Power9 (2.6/2.7 M txn/s, 9.5/9.8x). Then, these CC schemes gradually degrade similarly on all three hardware platforms until thrashing at 88 cores. Notably, this thrashing occurs when using two sockets on HPE but only one socket on Power9 and Power8, i.e. across NUMA distance 1 on HPE but NUMA-local on the Power platforms. This fact and similar abort ratios on all three platforms beyond the thrashing point indi-

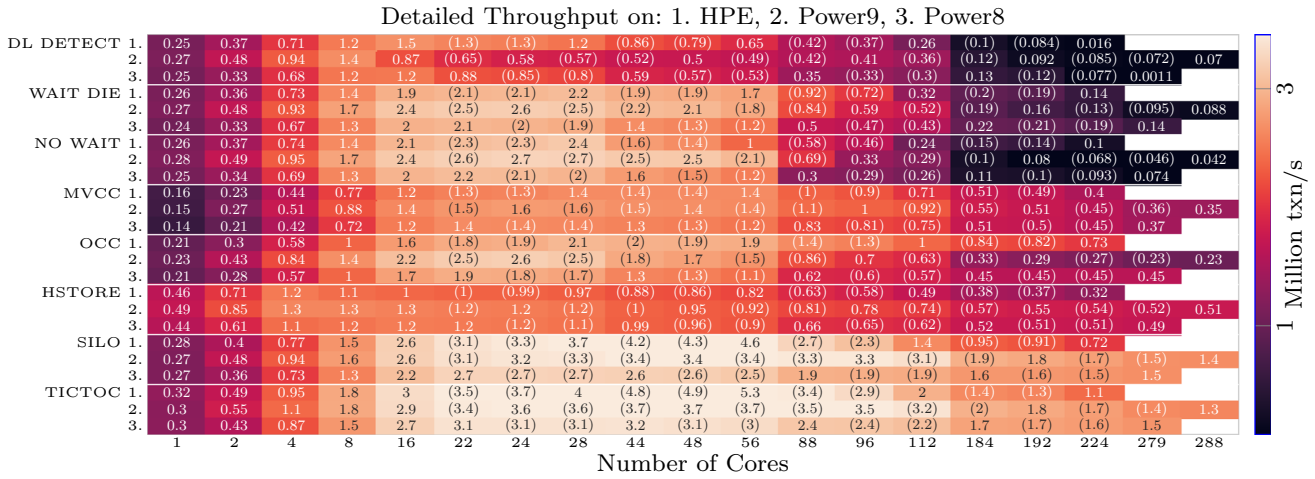


Fig. 12 Detailed throughput for TPC-C (Million txn/s) under high conflict on HPE, Power9, and Power8. The figure shows the speedup of different CC schemes (on y-axis) when scaling the number of cores (on x-axis). Per CC scheme, we have 3 rows — one for each platform (1. HPE, 2. Power9, 3. Power8)

cate overwhelming conflicts as cause for this thrashing of WAIT DIE and NO WAIT (rather than NUMA or other hardware properties). At high core counts, NUMA additionally takes effect. Then, WAIT DIE and NO WAIT benefit from lower NUMA latency on Power8, though only by less degrading.

Moving forward to the other schemes, we see further interesting behaviours: (1) MVCC and OCC again scale differently than the pessimistic locking schemes on larger core counts, peaking at 24 cores (on Power 9 with 1.6/2.6 M txn/s). Afterwards, OCC degrades less on HPE than on Power9 and Power8, resulting in significantly higher throughput on HPE at high core counts despite the stronger NUMA effect on this hardware platform, as we will see later. (2) HSTORE also reaches peak performance on Power9 with 1.35 M txn/s at 4 cores. Afterwards, its performance converges between Power9 and Power8. In contrast, HSTORE gradually falls behind on HPE between 4 and 56 cores (one full socket), then worse NUMA effects on HPE further slow down HSTORE. (3) Finally, SILO and TICTOC initially perform best on HPE, peaking at 56 cores with 4.6/5.3 M txn/s. Beyond this peak, SILO and TICTOC degrade steeply on HPE. Instead, on Power9 and Power8 their throughput scales worse with a lower peak but also less degrading than on HPE. Notably, the performance of both CC schemes drops at 88 cores on Power8 within a socket. Since Power9 does not exhibit such performance drop within a single socket, fewer hardware resources of the Power8 processor (especially L3 cache) and subsequent resource contention within SILO and TICTOC seem to cause the earlier performance drop.

Comparing the performance of the CC schemes for this high conflict workload reveals an influence of the hardware properties, e.g. some CC schemes react stronger to NUMA and cache contention than others. Overall, the pessimistic CC schemes degrade strongest at high core counts on all three hardware platforms. Notably, among the pessimistic CC schemes NO WAIT stays ahead until utilising all cores of a socket on the individual platforms, at which point WAIT DIE overtakes. This indicates cache contention and NUMA as factors strongly influencing the pessimistic CC schemes besides conflicts, i.e. the simpler NO WAIT is not only sensitive to conflicts in the workload but also to contention inside the hardware, whereas WAIT DIE copes better with higher conflicts and contention at the cost of overhead. A similar influence of hardware effects versus overhead can be observed between MVCC, OCC, and HSTORE. On Power9 and Power8 at higher numbers of cores with high conflict, HSTORE despite its coarse partition locking catches up with MVCC and OCC circumventing NUMA and resource contention effects due to the lower overhead, whereas the medium overhead OCC performs the best on HPE. Finally, SILO and TICTOC perform the best on all three platforms. Their peak performance is further ahead of the other CC schemes on HPE than on Power9 and Power8, but as NUMA takes effect SILO and TICTOC degrade less on the Power platforms.

Further analysis of the detailed time breakdowns[3] confirms that the hardware characteristics effect how the individual CC schemes spent time. The time breakdowns on Power8 reveal increased proportions of time spent for index accesses and concurrency control compared to HPE, on which more time is spent for actual work. Profiling confirms that latching within the CC schemes and index traversal are the hotspots on Power8, both of which are sensitive to memory latency. Notably, latching occurs to different extends in

[3] Figures omitted for brevity are available online [5].

the CC schemes and in different phases, i.e. during transaction execution categorised as CC Mgmt. or when committing transactions categorised as *Commit*. The observations for the individual CC schemes are accordingly. For example, the pessimistic locking schemes spend more time acquiring locks and committing, whereas MVCC and OCC spend more time committing. On Power9, the time breakdowns exhibit similar increases in time spent for index accesses and concurrency control, yet lower than on Power8. That is, the time spent for index accesses and concurrency control is related to the cache sizes of the hardware platforms, resulting in the least time spent on HPE with the largest L1 and L2 caches per logical core followed by Power9 with a larger L3 cache than Power8 (cf. Table 2b). With increasing core counts and accordingly more conflicts, these differences vanish as time spent for waiting or aborting dominates.

Insight Under high conflict, regardless the hardware platform no CC scheme utilises high core counts effectively, i.e. the CC schemes only initially scale well with increasing core counts, but quickly thrash under the overwhelming conflicts. Yet, their specific scaling behaviour indeed depends on the hardware, especially on processor characteristics (e.g. caches capacity) and NUMA, further analysed in the following sections.

4.1.2 Scaling on different real hardware—low conflict

In this second experiment, we determine how the different CC schemes scale on the different hardware platforms providing a high number of cores, when low conflict workload permits high concurrency. Accordingly, Fig. 13 shows the throughput of the CC schemes on HPE, Power9, and Power8. Briefly summarised, all CC schemes present positive scaling behaviour on all three hardware platforms, HSTORE initially performs the best, but most CC schemes follow HSTORE in a pack, and MVCC is behind at least for lower core counts.

However, a closer comparison between the three hardware platforms indicates again two interesting trends for this low conflict workload. First, the throughput of all CC schemes increases in distinctly different slopes on the three platforms, i.e. the hardware platforms seem to have a distinct effect on the scaling behaviour. Second, as the number of cores increases, the relative performance of the CC schemes distinctly differs between HPE and the two Power platforms. For detailed analysis, Fig. 14 shows comparisons, indicating for each CC scheme the speedup of one platform over another (e.g. Power8 vs. Power9) at the same number of cores.

The comparison of Power9 and HPE in Fig. 14 (Power9 vs. HPE) indicates that all CC schemes are faster on Power9. However, the speedup on Power9 compared to HPE varies in distinct pattern, corresponding to the increasing NUMA distance. For the pessimistic locking schemes, the throughput difference between Power9 and HPE shrinks until using 2

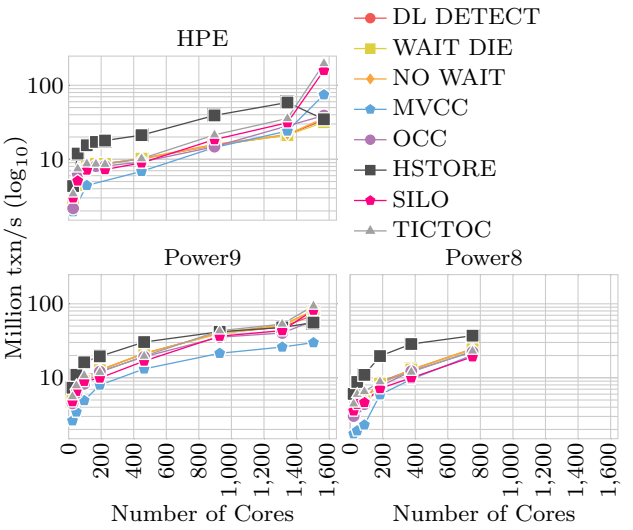


Fig. 13 Throughput for TPC-C under low conflict on HPE, Power9, and Power8

sockets (112 cores) on HPE, then throughput on HPE falls behind and with more than 224 cores across 4 sockets on HPE (across the farthest NUMA distance 3) throughput drops even further. The other CC schemes react similarly to these on HPE, except for HSTORE, which instead is affected by the closest boundary beyond one socket and farthest boundary above 4 sockets (with NUMA distances 1 and 3). Further, the closest boundary after 56 cores on HPE has a diverse effect on the CC schemes. This NUMA boundary only has a negative effect on the fastest two CC schemes (i.e. HSTORE and TICTOC) as well as OCC. Instead, the other CC schemes scale well past one socket (56 cores) on HPE and in fact close in onto the throughput on Power9.

Comparing Power8 and HPE in Fig. 14 (Power8 vs. HPE), indicates the same effect as observed in comparison with Power9. Also on Power8, the performance of all CC schemes initially is ahead; then, their performance on HPE catches up around 56 - 112 cores (across two sockets with NUMA distance 1). In fact, HPE overtakes Power8, on which the CC schemes struggle due to resource contention (to be discussed in Sect. 4.2.1). Remarkably, the larger processor resources on HPE compensate for its worse NUMA properties (lower bandwidth and higher latency), when operating across 2 sockets. Across more than 2 sockets (112 cores), the performance of most CC schemes on HPE is even to Power8. Only HSTORE and MVCC straggle on HPE, due to their higher load on the memory subsystem (i.e. sheer performance of HSTORE and overhead of MVCC).

The comparison of Power9 and Power8 in Fig. 14 (Power 9 vs. Power 8 or vice versa) indicates improved performance of the Power9 processor over Power8, as throughput on one socket is 1.2 - 2x higher. Notably, beyond one socket the performance benefit of Power9 stagnates or even decreases.

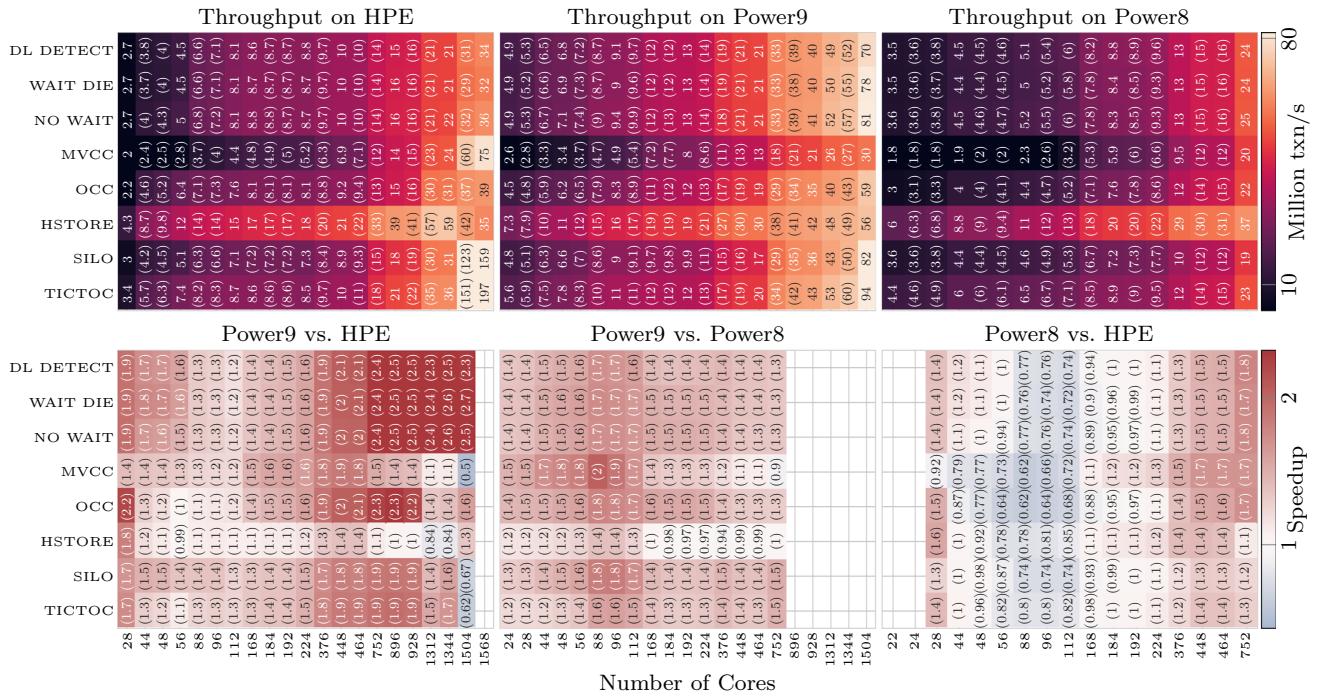


Fig. 14 Detailed throughput for TPC-C under low conflict on HPE, Power9, and Power8 (1st row) and comparison between the hardware platforms (2nd). The comparisons indicate by which speedup ratio the throughput differs for a CC scheme (on y-axis) at the same number of cores (x-axis) on one platform versus another platform

Consequently, the strong NUMA topologies of both Power platforms similarly boost concurrency control at large scale. This confirms a general advantage of Power's stronger NUMA topology and importantly indicates the relevance of NUMA properties for the performance of concurrency control.

Furthermore, the time breakdowns[4] indicate diverging internal behaviour of the CC schemes on the hardware platforms. Similar to the high conflict workload, on Power8 the CC schemes spend significant time for concurrency control and index accesses, while on HPE for useful work (e.g. accessing records). Also Power9 shows increased time spent for concurrency control and index accesses, but again overall lower than on Power8 and biased towards index accesses (less for concurrency control but more for index accesses). Profiling on Power8 confirms this continued trend, again identifying latching as bottleneck related to memory latency. Conversely, for HPE, these observations hint at memory bandwidth as bottleneck for this low conflict workload.

Regarding the second trend about the relative performance of the CC schemes, on Power9 and Power8 the pessimistic locking schemes perform better than on HPE. Notably, these perform better than SILO and TICTOC for 96 - 1504 and 192 - 928 cores, respectively. Also, OCC improves but only at larger core counts and not as much as the pessimistic schemes. Consequently, on Power, OCC overtakes SILO, but falls behind the pessimistic schemes. In contrast, MVCC provides the worst throughout on Power with a growing gap to the other CC schemes, whereas on HPE MVCC does overtake the other CC schemes at large scale. These differences of the relative performance of the CC schemes indicate two underlying causes for this second trend: (1) Especially the latency sensitive pessimistic locking schemes benefit from the lower latency in Power's NUMA topology; (2) Resource intense CC schemes (e.g. MVCC) benefit from the larger hardware resources of the processors in HPE.

Insight Under low conflict, the NUMA characteristics of the specific hardware platforms clearly affect the performance of the CC schemes, i.e. the scaling slopes of the CC schemes closely match the NUMA topology. The CC schemes generally benefit from lower latency and higher bandwidth in the NUMA topology. Yet the individual CC schemes benefit differently from either better latency or bandwidth and resource contention within the processor influences their scaling behaviour.

4.2 Zooming into hardware aspects

Having identified diverging behaviour on the different hardware platforms, we now zoom into those aspects that realise

[4] Figures omitted for brevity are available online [5].

the large number of cores: (1) Hardware parallelism within the processors and (2) the topology connecting processors in a single system.

4.2.1 Simultaneous multithreading

The superscalar processors of today's hardware employ several techniques to implement hardware parallelism. Besides a high number of physical cores, the processors also employ (superscalar) instruction-level parallelism (ILP) [20] and Simultaneous Multithreading (SMT) [10]. SMT establishes multiple parallel execution streams as logical cores to better utilise the resources of their underlying superscalar physical core, especially to facilitate thread-parallel software such as OLTP DBMSs.

While many of today's superscalar processors employ these general techniques, the specific implementations differ [1,29–31]. Especially the Power processors utilise sophisticated SMT with a high degree of parallel execution streams on a smaller number of physical cores, up to 8 such streams (i.e. SMT-8) [30,31]. Notably, from Power8 to Power9 IBM's hardware designers have enhanced the SMT implementation, e.g. with advanced scheduling of the execution streams. In contrast, Intel processors mainly drive hardware parallelism by the number of physical cores and use simpler SMT with two parallel execution streams (SMT-2) [29].

These elaborate techniques of hardware parallelism depend on processor resources and the software as well as the workload running on top. Therefore, our particular questions are how big this benefit can be as OLTP workloads typically strain the memory subsystem more than other processor resources and if the CC schemes allow for sufficient concurrency to utilise the parallel hardware execution streams of SMT.

In the following, we analyse the benefit of SMT for OLTP workloads, focusing on the sophisticated and high-degree SMT (up to SMT-8) of the Power processors. In the experiments, we use all physical cores of a single processor and observe the throughput for increasing SMT degree. We first analyse the best-case benefit using the low conflict TPC-C workload, before also considering high conflict scenarios.

High SMT degree for low conflict OLTP Figure 15 shows the throughput for the low conflict TPC-C workload of all CC schemes under increasing SMT degree and the speedup relative to SMT-1. On the Power9 processor, most CC schemes speedup equally with increasing SMT degree, despite differing throughput; 1.7 - 1.8x for SMT-2, 2.4 - 2.5x for SMT-4, and 3.3 - 3.4x for SMT-8. Only HSTORE utilises SMT better with a speedup of 2.6x for SMT-4 and 3.9x for SMT-8, relating to low overhead and exceptional performance for low conflict workloads. Notably, despite a significant speedup of all CC schemes, the speedup of SMT on the Power9 processor is sublinear for this low conflict OLTP workload.

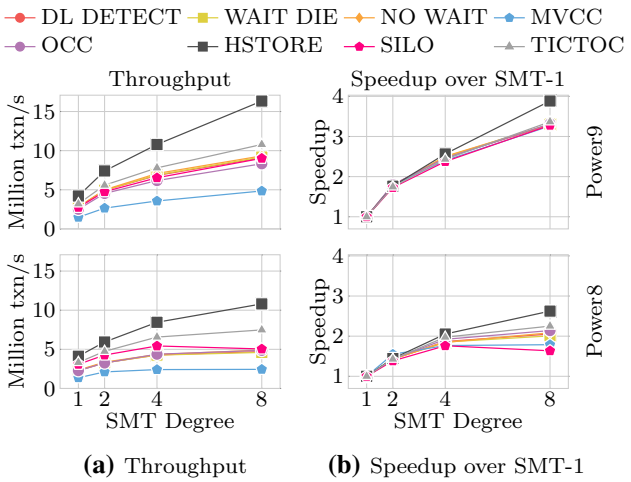


Fig. 15 Effect of broad SMT in Power9 and Power8 processors on throughput for TPC-C under low conflict

On the Power8 processor, in contrast, the CC schemes achieve overall lower throughput and speedup than on Power9 (SMT-2: 1.4 - 1.5x, SMT-4: 1.8 - 2.1x, SMT-8: 1.6 - 2.6x). That is, SMT of the Power8 processor provides less benefit and the speedup of the CC schemes also diverges, in three distinct groups. (1) HSTORE utilises SMT best with the highest speedup, as on Power9. (2) TICTOC, OCC, and the pessimistic locking schemes follow with still positive speedup for SMT-8, but progressively less in according order. (3) The speedup of MVCC stagnates from SMT-4 and for SILO even decreases from 1.8x for SMT-4 to 1.6x for SMT-8.

Notably, the three groups with distinct benefit of SMT comprise CC schemes with similar memory footprints and the speedup of these groups correlates with these footprints, i.e. the group of CC schemes with the smallest footprint gains most speedup from SMT and inversely the group with the largest footprint gains least. This correlation to the memory footprint and the increasing gap to Power9 indeed indicates increasing resource contention for SMT on Power8. Comparing their cache capacity highlights the larger L3 cache per logical core on Power9 (cf. Table 2b) [30,31]. Evidently, sufficient L3 cache capacity for all the execution streams is an important factor to effectively utilise SMT.

On the Intel processor with only SMT-2, we make similar observations, omitted from Fig. 15 due to the small SMT degree. For example, SMT-2 of that Intel processor provides a speedup of 1.5x for TICTOC from a throughput of 5.25 M txn/s with SMT-1 to 7.92 M txn/s with SMT-2.

Insight Overall, SMT indeed benefits our favourable (i.e. low conflict) OLTP workload, yet with sublinear speedup in relation to the SMT degree. The sophisticated SMT of the Power9 processor provides broad benefit for all CC schemes up to the highest SMT degree (SMT-8). In contrast, resource

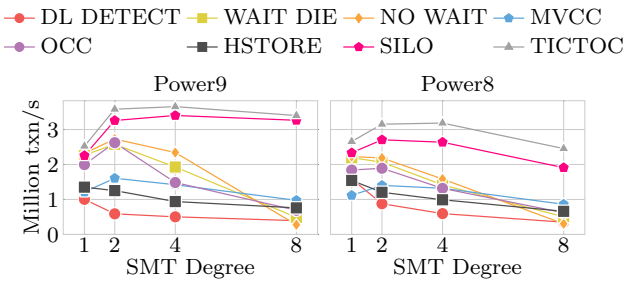


Fig. 16 Throughput of broad SMT in Power9 & Power8 processors for TPC-C under high conflict

contention limits the benefit of SMT on the Power8 processor, indicating a dependency between the benefit of SMT and the resource footprint of the CC schemes.

High SMT degree for high conflict OLTP For the second workload with high conflict, throughput of the CC schemes under increasing SMT degree and speedup relative to SMT-1 is shown in Fig. 16. Overall, the CC schemes barely benefit from SMT on neither Power9 nor Power8. In detail, on Power9, SILO and TICTOC utilise SMT best. These speed up by 1.4*x* with SMT-2 and maintain this speedup for SMT-4 and SMT-8. In contrast, the remaining CC schemes speed up with SMT-2 by a smaller factor — if at all. Latest with SMT-4, their speedup declines to a slowdown (<1x speedup). DL DETECT and HSTORE immediately slow down with SMT-2 (0.59*x* and 0.93*x*, respectively). On Power8, the CC schemes benefit even less from SMT, i.e. the speedup for SMT-2 is lower and for higher SMT degrees the slowdown is stronger. Notably, MVCC has the same speedup on both Power9 and Power8, throughput is higher on Power9 by stable 10%. In conclusion, conflicts are the determining factor for the performance of all CC schemes and prohibit general benefit of SMT. Yet, the improved SMT of Power9 is still noticeable, albeit more limited than under low conflict.

Insight For high conflict OLTP workload, the performance of all CC schemes is widely determined by the conflicts rather than SMT, yet some benefit of SMT appears, especially from the Power9 processor.

4.2.2 Non-uniform memory access

Today, thousands of cores are only available via multi-socket hardware imposing the Non-Uniform Memory Access (NUMA) effect for memory accesses. Such multi-socket hardware connects its processors (and memory) in a tiered non-uniform topology, through which the processors communicate and mutually access memory. As the topology connecting the processors is tiered and non-uniform, so are the performance characteristics for processors when communicating or accessing memory, i.e. bandwidth and latency between processors in the topology differ. These diverging

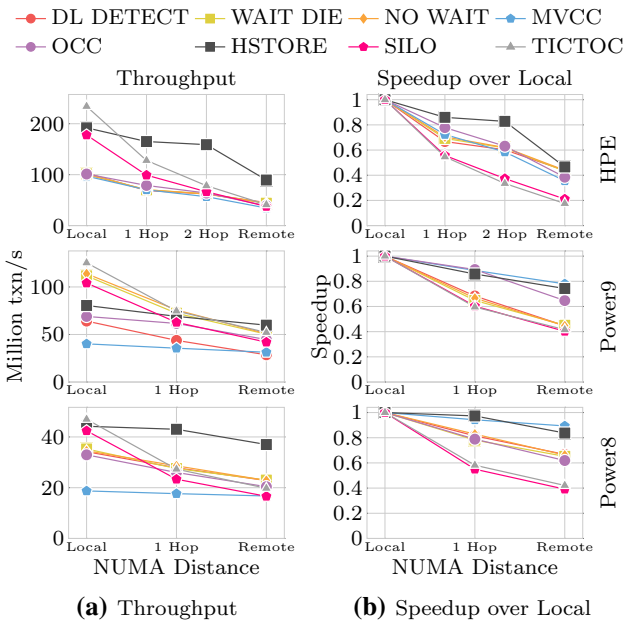


Fig. 17 Performance for TPC-C under low conflict with strict access to home warehouse at specified NUMA distance (on x-axis) on HPE, Power9, and Power8. Speedup is reported as ratio of throughput at the specified NUMA distance over *Local*. For Power, the distance *2 Hop* is missing since this hardware has a shallower topology than HPE (cf. Sect. 2.2)

performance characteristics of the underlying hardware (the NUMA effect) impact the performance of a DBMS depending on its communication and memory access pattern.

In the following, we analyse the NUMA effect of our three hardware platforms on the CC schemes, which all employ different technologies and topologies to connect their processors (see Sect. 2 for more details). For this analysis, we start with an extreme scenario isolating the NUMA characteristic of the three hardware platforms and their effect on the CC schemes when all memory accesses have a predefined NUMA distance. In a second experiment, we compare the NUMA effect on the CC schemes using a more realistic and complex scenario with NUMA effects imposed by the workload.

Isolated NUMA effects (fixed distance) First, we analyse the NUMA effect on the CC schemes in an extreme scenario, where transactions strictly access memory at a fixed (specified) NUMA distance. This extreme scenario overall exposes the differing NUMA characteristics of our hardware platforms and subsequently reveals their influence on the CC schemes. For this scenario, we restrict the TPC-C transactions to only access their home warehouse and allocate this warehouse on memory with the specified NUMA distance. Further, we use the low conflict workload and the maximum cores, isolating the effect of operating across a specified NUMA distance from other effects (e.g. conflicts).

Figure 17 shows the throughput and speedup of the CC schemes under increasing NUMA distance on HPE, Power9, and Power8. Since the maximum number of cores (where the NUMA effect is strongest) differs on the hardware platforms and thus the throughput, we rather focus on the speedup of the NUMA distances 1 - 3 (*1 Hop, 2 Hop, Remote*) over the local NUMA distance 0 (i.e. when all data are accessed on the local NUMA region/processor), as shown in Fig. 17b. Overall, as expected the NUMA effect (deteriorating bandwidth and latency) indeed degrades performance as the NUMA distance increases. Yet, throughput and speedup of the CC schemes show several trends on the different hardware platforms.

On HPE when accessing only local memory, the CC schemes HSTORE, SILO, and TICTOC achieve remarkable throughput of 178 - 234 M txn/s. However, when accessing farther memory, SILO and TICTOC immediately slow down sharply by 0.56x and 0.55x at NUMA distance 1, respectively. Then, SILO and TICTOC slow down at a lower rate, to 0.21x (38 M txn/s) and 0.18x (41 M txn/s) for NUMA distance 3 (Remote). On Power9 and Power8, SILO and TICTOC slow down similarly for the NUMA distance 1 (0.55 - 0.6*x*), but for the farthest NUMA distance (Remote) their slowdown is more graceful (0.39 - 0.42*x*). In contrast, HSTORE slows down much less on all three hardware platforms. For NUMA distances 1 - 2, HSTORE slows down least on Power8 (0.97*x*), followed by HPE and Power9 on par (0.86*x*). Afterwards, the farthest NUMA distance affects HSTORE again the least on Power8 (0.84*x*), followed by Power9 (0.74x), but HPE falls behind (0.47*x*).

The remaining CC schemes generally slow down more gracefully under increasing NUMA distance. Notably, on HPE, the pessimistic locking schemes (DL DETECT, WAIT DIE, and NO WAIT initially slow down stronger for NUMA distance 1 (0.67 - 0.7*x* vs. 0.72 - 0.78*x*) but then slow down at a lower rate, while MVCC and OCC slow down stronger at the farthest NUMA distance 3 (Remote). On Power9, the pessimistic locking schemes slow down similarly. On Power8, however, these CC schemes slow down less, i.e. by 0.78 - 0.83x for distance 1 and 0.65 - 0.67*x* for distance 3. That is, the lower latency in the topology of Power8 significantly benefits the pessimistic locking schemes.

In contrast, MVCC slows down most on HPE, with Power9 and Power8 similarly ahead for NUMA distance 1 (HPE: 0.72*x*, Power9: 0.89*x*, Power8: 0.94*x*), but at the farthest NUMA distance Power9 falls behind and Power8 leads again (HPE: 0.36*x*, Power9: 0.74*x*, Power8: 0.89). From the higher bandwidth of the Power platforms and in turn higher bandwidth in Power8 than Power9, we conclude that MVCC benefits from higher bandwidth in the topology (and lower latency).

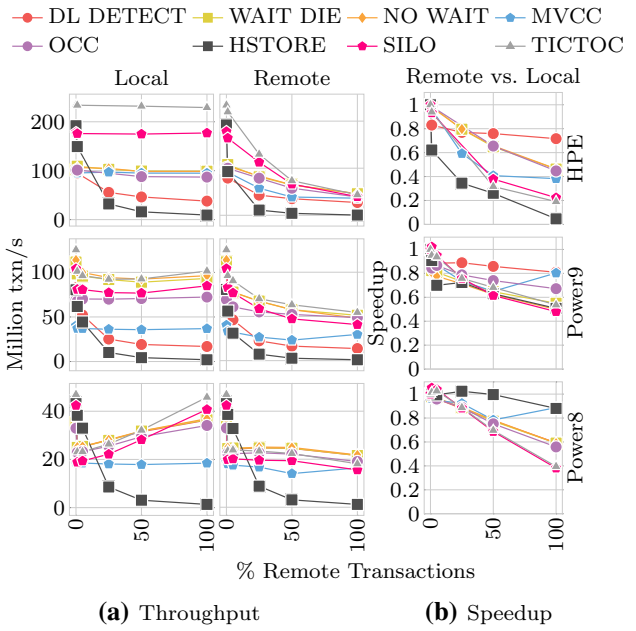
Finally, OCC also presents diverse slowdown across the three hardware platforms. Initially, at NUMA distance 1 OCC slows down least on Power9, followed by HPE and Power8 on par (Power9: 0.89*x*, HPE: 0.78*x*, Power8: 0.79*x*), while at the farthest NUMA distance 3, Power9 and Power8 are equally ahead of HPE (Power9: 0.65*x*, Power8: 0.62*x*, HPE: 0.38*x*).

Insight Overall, two notable trends appear relating to the NUMA characteristics of the three hardware platforms. First, the latency sensitive pessimistic locking schemes do best on Power8 providing the lowest latency in its topology. On HPE and Power9 instead, which have similarly higher latencies than Power8 for *NUMA Remote* accesses, these schemes perform similarly worse. Second, CC schemes that require more bandwidth, either due to sheer performance as for HSTORE or due to memory overhead as for MVCC, perform better on Power9 and Power8, both of which provide higher-bandwidth interconnects in their topologies. Both these trends confirm our early observations of NUMA effects on the scaling behaviour of the CC schemes on the three hardware platforms.

Workload imposed NUMA effect The previous experiment highlights effects on the CC schemes relating to the NUMA characteristics in an extreme scenario. However, realistic operating conditions of OLTP DBMSs are more complex. On one hand, DBMSs commonly attempt to mitigate extreme NUMA effects by strategies like NUMA-aware database partitioning. On the other hand, realistic workload dictates the access pattern, still imposing NUMA effects (and other non-NUMA effects). We now analyse these more realistic workload-imposed NUMA effects using TPC-C's remote transactions. That is, TPC-C is commonly partitioned by warehouses (also in our experiments) mitigating NUMA effects. Yet, TPC-C specifies so-called remote transactions that apart from their home warehouse span further warehouses (remote warehouses), thus these remote transactions are not partitionable and cause workload-imposed NUMA effects.

In the following experiment, we use the combination of following two setups to isolate the NUMA-related from the other non-NUMA effects in this more complex scenario: (1) In the first setup, we analyse the non-NUMA related effects of remote transactions. For this, we vary the amount of remote transactions across warehouses but use only local memory for all warehouses (i.e. no NUMA effects occur); (2) In the second setup, we then distribute warehouses across NUMA regions and thus observe the combined (NUMA and non-NUMA) effects imposed by remote transactions. Consequently, we can thus better isolate the NUMA-related from the non-NUMA-related effects on the CC schemes by comparing their performance in these two settings.

In detail, for setup *(1) NUMA Local*, we allocate the remote warehouses on local memory (alongside the home warehouse and transaction executor). In contrast, for setup *(2) NUMA Remote*, we allocate the remote warehouses the



(a) Throughput **(b)** Speedup

Fig. 18 Effect of workload-imposed NUMA by increasing ratio of remote transactions (on x-axis) on TPC-C under low conflict in the two different setups (*Local* and *Remote* as described in the setup of this experiment) and the three platforms: HPE, Power9, and Power8. Figure (a) shows the throughput for increasing ratio of remote transactions. Figure (b) shows the ratio of throughput of the *Remote* over the *Local* setup

farthest away from the transaction executors, i.e. remote warehouses are at remote NUMA distance 3 but the home warehouses remain at local NUMA distance 0. The remaining setup is identical to the prior experiment (cf. *Isolated NUMA effects*).

Figures 18a shows the performance in the two described settings (*(1) Local* and *(2) Remote*), when transactions increasingly access remote warehouses (% remote transactions) either on (1) local memory or (2) remote memory. In addition, Fig. 18b compares the performance in these two settings (*Remote vs. Local*) for the same ratio of remote transactions. We first analyse how the different CC schemes are affected by the aforementioned effects focusing first on the HPE platform. In a second step, we then compare the effects across the different hardware platforms to identify the effect of their NUMA characteristics.

Starting with the CC schemes on HPE (top row of Fig. 18): while most CC schemes provide stable throughput for the *Local* setting on HPE, they all degrade in the *Remote* setting due to the NUMA effect (also on the Power platforms, though with further effects which we discuss later). Notably, DL DETECT and HSTORE significantly degrade already in the *Local* setting without the NUMA effects, i.e. non-NUMA effects impact these CC schemes as well.

Figure 18b shows the performance ratio when increasing the NUMA distance for remote warehouses in setup (2) compared to the *(1) Local* setup. This provides a more detailed insight into the effects of remote transactions. Overall, the resulting effects on CC schemes can be grouped into three categories:

1. DL DETECT drops to 0.83x at 1% remote transactions but then degrades only to 0.72x, which indeed is the least effect across all CC schemes. Consequently, conflicts (and other non-NUMA effects) affect DL DETECT more than NUMA.
2. Conversely, the other pessimistic CC schemes (WAIT DIE and NO WAIT) as well as OCC, SILO, and TICTOC suffer more from the NUMA effects. These significantly slow down with *NUMA Remote* compared to *NUMA Local*, while their throughput for NUMA Local is mostly stable.
3. HSTORE and MVCC suffer from the combination of NUMA effects and non-NUMA-related conflicts. While for HSTORE a combined effect relates to its high sensitivity to conflicts (as observed previously), for MVCC an additional effect of conflicts only appears by comparison with the prior experiment on NUMA effects (cf. Fig. 17). Previously MVCC suffered less NUMA effects, when there were no conflicts in the workload. Consequently, the conflicts indeed amplify the NUMA effect for MVCC.

Comparing the hardware platforms (2nd and 3rd row of Fig. 18), we see that the CC schemes on the Power platforms behave similar to HPE. However, looking into the detailed behaviour, we see that the workload-imposed NUMA effects depend on the individual NUMA characteristics of the hardware platforms. For example, in our following analysis we confirm the advantage of the better NUMA characteristics of the Power platforms compared to HPE, providing more stable behaviour (as already observed in the previous experiment). This can be seen by the fact that the CC schemes on the Power platforms for the *Remote* setup in Fig. 18 (right column) show a shallower drop when compared to HPE. In the following, we now discuss the details that lead to this behaviour.

In Fig. 18a, the throughput of the CC schemes for *NUMA Remote* degrades depending on three factors: the sensitivity of the CC schemes to NUMA, the NUMA characteristics of the specific hardware platform, and non-NUMA effects such as cache pollution. These effects appear as follows on the three hardware platforms for the CC schemes previously categorised as significantly affected by NUMA (and insignificantly by non-NUMA effects): (1) On HPE, as already determined, the NUMA effect strongly and continuously degrades the CC schemes; (2) on Power9, the better NUMA characteristics degrade the CC schemes less, but the smaller cache causes a small drop for 1% remote transactions; (3) on Power8, the small cache causes a significant non-NUMA-related drop for 1% remote transactions for both *NUMA*

Remote and *NUMA Local*, afterwards the CC schemes also degrade due to the NUMA effect similar to Power9, as detailed in Fig. 18b.

Finally, the CC schemes of the other categories (not mentioned above) also diverge between the three platforms. We summarise the most important findings for those schemes in the following. Figure 18b indicates that the cache pollution on Power8 exposes DL DETECT to NUMA effects, as there is no NUMA effect on HPE or Power9. Furthermore, observing the speedup of *Remote vs. Local* in Fig. 18b confirms for the CC schemes of the third category (e.g. HSTORE) that the NUMA effects are amplified by non-NUMA effects. As the NUMA characteristics improve from HPE to Power9, and further from Power9 to Power8, we observe that the speedup of *Remote vs. Local* converges towards 1*x*, i.e. the performance of these CC schemes indeed becomes independent of NUMA effects and depended on the other non-NUMA effects.

Insight In the more realistic scenario of workload-imposed NUMA effects (by TPC-C remote transactions), the CC schemes not only face NUMA effects but also other effects. To summarise, we have seen that they are affected in three groups: (1) one group is mainly affected by non-NUMA effects (e.g. conflicts), such as DL DETECT, (2) another group is primarily affected by NUMA effects, e.g. WAIT DIE or TICTOC, and (3) the last group is affected by the combination of NUMA effects and conflicts, e.g. MVCC and HSTORE. These findings apply to all three hardware platforms, in variations according to the specific hardware characteristics as previously observed for the isolated NUMA effect.

4.3 The full TPC-C benchmark

In the previous experiments, we observed a significant impact of conflicts and data locality on the behaviour of the CC schemes. However, besides conflicts and data locality, the type of workload and operations is a major aspect. Therefore, in this final evaluation step, we analyse the effect of the workload on the CC schemes in more detail. In particular, we evaluate the contrast between a more comprehensive workload covering the full TPC-C transaction mix (all 5) versus the often used more narrow transaction mix comprising just the NewOrder and Payment transactions, which was used in the simulation of prior work [71]. Notably, the full transaction mix includes read-heavy and additionally more expensive (i.e. longer-running) transactions, such as Stock-Level aggregating records from many districts. In addition, the full mix requires additional indexes increasing the cost of the NewOrder and Payment transactions (used in the more narrow mix) as well.

In the following, we again start with an analysis of the high conflict workload and then discuss the results for the low conflict workload.

4.3.1 Full TPC-C under high conflict

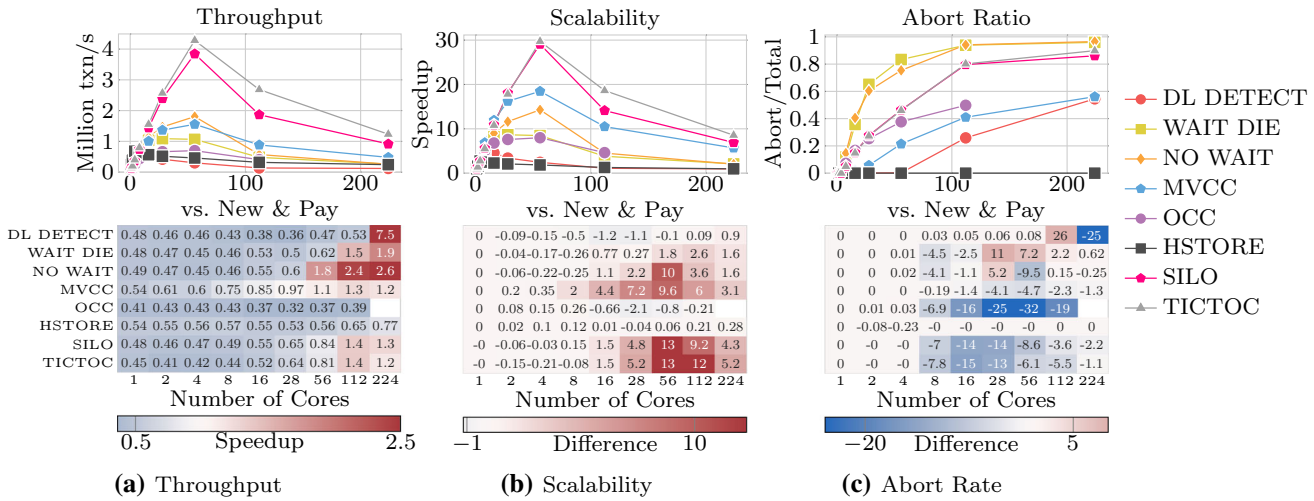
In this experiment, we analyse how the behaviour of the CC schemes differs between the full and the narrow transaction mixes for high conflict workload. Most notable, the read-heavy transactions of the full mix are expected to affect the CC schemes depending on their ability to handle read-write conflicts. In a first step, we thus focus on diverging behaviour of the CC schemes between these two transaction mixes on the same hardware platform. Then, we assess whether their behaviour further differs across the hardware platforms. As in our previous experiments, we first evaluate the CC schemes on HPE and then compare the Power platforms.

Full vs. narrow mix on HPE Fig. 19 displays the performance of the individual CC schemes for the full TPC-C transaction and a comparison with the narrow mix (only NewOrder and Payment transactions), cf. Fig. 13. The top row provides an overview over the performance of the individual CC schemes. Overall, it shows that the CC schemes scale well initially, but eventually all thrash due to overwhelming conflicts—a similar behaviour as with the narrow transaction mix.

However, the comparison with the throughput of the narrow mix (Fig. 19a, 2nd row) indicates broadly worse throughput with the full mix until about 56 cores. At higher core counts, though most CC schemes indeed provide better throughput (e.g. NO WAIT at 224 cores 2.6*x* over the narrow mix). Remarkably, the CC schemes better handle increasing conflicts and NUMA effects with the more (read)-intense transactions in this full mix. Only HSTORE does not quite close the performance gap between the full transaction mix and the narrow mix (0.53 - 0.77*x* the performance of the narrow mix) and OCC's performance for the full mix remains low at 0.4*x*, not improving at higher core counts.

The detailed scaling behaviour in Fig. 19b indeed indicates that this positive effect of the heavier transactions in the full transaction mix already starts at lower core counts. The comparison between the full and the narrow mix (Fig. 19b, 2nd row) shows that already from 8 cores the CC schemes exhibit better scaling for the heavier transactions, though beyond 56 cores (more than one socket) the lead decreases. Notably, SILO and TICTOC benefit the most (peak improvement), while MVCC benefits across the widest number of cores.

Having identified diverging impact of the full TPC-C transaction mix on the CC schemes, we now analyse the causes in further details. Specifically, we search for (1) reasons reducing the performance at lower core counts as well as improving the performance at higher core counts and (2) reasons for higher impact on some CC schemes than others.



(a) Throughput **(b)** Scalability **(c)** Abort Rate

Fig. 19 Throughput, scalability, and abort ratio for full TPC-C transaction mix under high conflict on HPE. In the 2nd row, throughput and scalability, and abort rate are compared between the full and the narrow TPC-C mix (only NewOrder & Payment). For throughput (a, 2nd row), we compare the speedup of the full over the narrow mix. Since scalability as such is reported as speedup over 1 core (b, 1st row), we rather compare the scalability as difference (b, 2nd row) between the speedup of the full mix minus the narrow mix. Likewise, the abort ratio (c, 2nd row) is compared by the difference, i.e. abort ratio of the full mix minus the abort ratio of the narrow mix. In all plots, one data point for OCC (at 224 cores) is missing, since here the OCC scheme "froze" due to high conflicts

For the first case, as the full transaction mix introduces additional read-write conflicts and longer transactions, there are two major differences between the full and the narrow transaction mix potentially causing the observed general divergence: Conflict handling and amount of actual work. If conflict handling has a major influence on the observed throughput and scaling behaviour, then the CC schemes should exhibit similarly diverging abort rates. However, in Fig. 19c the abort rates for the full mix and the comparison with the narrow mix are ambiguous without a clear effect of the heavier transactions. For example, MVCC has similar abort rates for both transaction mixes while throughput significantly differs. Similarly, the improved throughput for the full mix of SILO and TICTOC does not relate to their abort rate. Consequently, the abort rates of the CC schemes surprisingly do not relate to their diverging throughput for the two transaction mixes.

As further step in analysing the impact of read-write conflicts and longer transactions, we analyse the time breakdowns [5] detailing how the CC schemes spend their time processing transactions of the full and the narrow mix (e.g. useful work, aborting, etc., cf. Table 3 in Sect. 3). The time breakdowns reveal that lower throughput for the full transactions mix relates to an increase in relative time spent for concurrency control in all CC schemes (i.e. CC Mgmt. or Commit), either in addition to increased waiting/aborting (for DL DETECT, WAIT DIE, NO WAIT, and OCC) or exceeding a reduction in waiting/aborting (for MVCC, SILO, and TICTOC). As the number of cores increases and throughput improves for the full mix, the time spent for concurrency control converges between the full and the narrow mixes. Instead, the time breakdowns of the full mix indicate a slight reduction in time spent waiting or aborting in conjunction with a slight lead in useful work. Consequently, the higher transaction throughput in the full mix relates to lower conflict at higher core counts. These two trends in the time breakdown imply that, first, the lower throughput for the full mix is not only associated with conflicts, but also with the higher load of the heavier transactions, making concurrency control more costly for all CC schemes compared to the narrow transaction mix. Second, at high core counts the heavier transactions dampen the impact of conflicts, allowing higher throughput especially for those CC schemes that can efficiently handle read-heavy transactions. This is not the case for DL DETECT, OCC, and HSTORE, as explained below.

The time breakdowns also provide insight into why DL DETECT, OCC, and HSTORE behave inconsistently with the other CC schemes, i.e. with increasing core counts these do not benefit (as much) from the heavier transactions. DL DETECT spends much more time waiting with the full transaction mix compared to the narrow mix, since waiting itself becomes more costly for DL DETECT due to traversing larger wait-for-graphs. OCC is initially slower due to more costly concurrency control like the other CC schemes, but at higher core counts aborting in OCC appears as new bottleneck. Remarkably, the time spent aborting increases for OCC despite lower abort rate for the full mix, i.e. for OCC aborting the heavier transactions is more costly and overshadows lower conflict. In contrast, HSTORE spends its time very similar for both transaction mixes, i.e. waiting time

eventually dominates as conflicts overwhelm HSTORE's partition-based locking regardless the type of work. Consequently, the performance of HSTORE converges between the two transaction mixes due to similarly dominating waiting time. In contrast to the prior three CC schemes, MVCC performs exceptionally better with the full mix and indeed it spends more time for actual work and less for aborting or waiting, confirming its ability to prevent read-write conflicts (similar applies to TICTOC and SILO).

Insight Under high conflict, the heavier transactions of the full TPC-C transaction mix make concurrency control of all CC schemes more costly. However, at large scale, these heavier transactions also dampen the impact of conflicts, especially benefiting CC schemes that efficiently handle read-write conflicts.

Power vs. HPE: Figure 20 displays the throughput of the CC schemes for the full TPC-C transaction mix on Power9 and Power8. Additionally, this figure provides a comparison with the narrow mix on these hardware platforms and the difference to the comparison with the narrow mix on HPE. The behaviour of the CC schemes for the full transaction mix on Power8/9 broadly resembles their behaviour on HPE. The most noticeable difference is that throughput is generally lower, i.e. the heavier transactions reduce throughput on Power8/9 more than on HPE. Accordingly, at low core counts the full mix lags further behind the narrow mix and at high core counts it is less ahead on the Power platforms.

The general cause for the slowdown for the full transaction mix on Power is the same as on HPE, i.e. especially at low core counts the heavier transactions make concurrency control more costly. Furthermore, the following three differences between the Power and the HPE hardware platforms stand out:

1. As the number of cores increases on Power, especially the pessimistic locking schemes benefit far less from the full mix compared to HPE. HSTORE even degrades on Power9 and Power8, with increasing numbers of cores it increasingly falls behind its throughput for the narrow mix. That is, these CC schemes as well as SILO and TICTOC deviate further apart from their performance for the narrow transaction mix as the number of cores increases on Power. For the pessimistic locking schemes, the time breakdowns reveal more time spent aborting rather than waiting. The time breakdowns of HSTORE, SILO, and TICTOC are very similar on the three hardware platforms, i.e. their behaviour is the same, but hardware performance makes the difference.
2. In contrast to the prior CC schemes, OCC copes better with the full mix on Power than on HPE. On Power9, OCC speeds up by 2.5x over the narrow mix, while on HPE it slows down by 0.4x. On Power8, OCC only reduces the performance gap, reaching 0.84x speedup at the maximum of cores.
3. Only MVCC does not diverge further, providing constantly less speedup ($-0.10x$) on Power than on HPE. Indeed, the time breakdowns of MVCC are similar for all three hardware platforms, i.e. its behaviour is the same, but hardware performance differs.

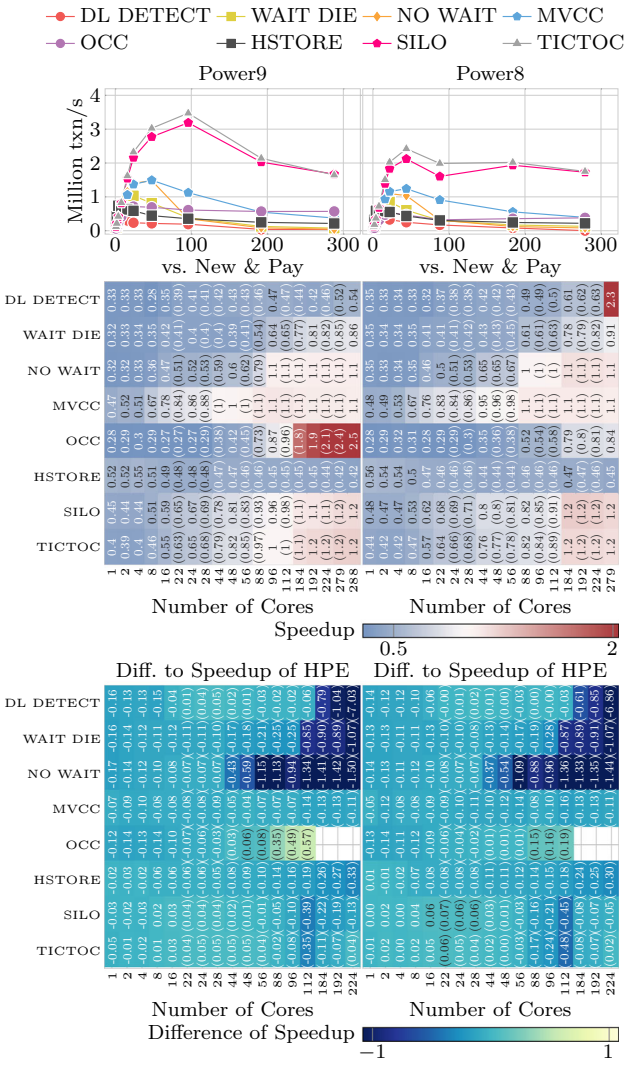


Fig. 20 Throughput for full TPC-C mix under high conflict on Power9/8 with comparison to narrow mix (NewOrder & Payment) and in 3rd row difference to comparison of full vs. narrow mix on HPE (cf. Fig. 19a)

Insight In conclusion, TICTOC and SILO handle high amount of conflicts best, regardless the hardware or workload type (full and narrow TPC-C mix), while MVCC proves its conflict handling advantageous for (read-)heavy workload (full TPC-C). Moreover, the specific performance of the CC schemes for heavier transactions as in the full TPC-C mix depends on the underlying hardware and the number

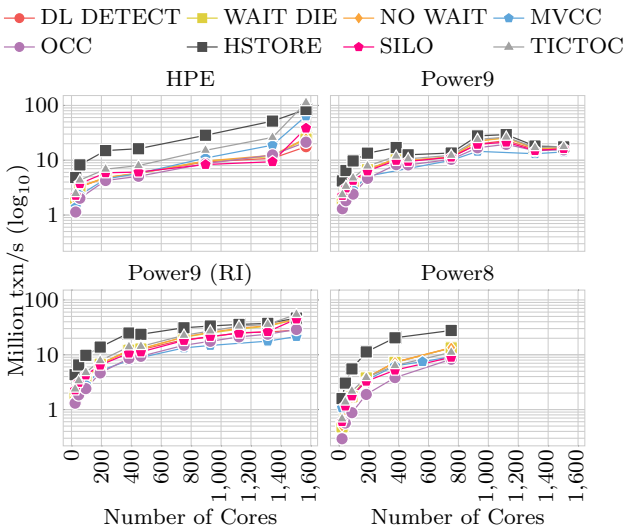


Fig. 21 Throughput of optimised DBx1000 for full TPC-C transaction mix under low conflict on HPE, Power9, and Power8. The full mix causes a performance bug on Power9, which is removed in "Power9 (RI)" by replicating internal data structures

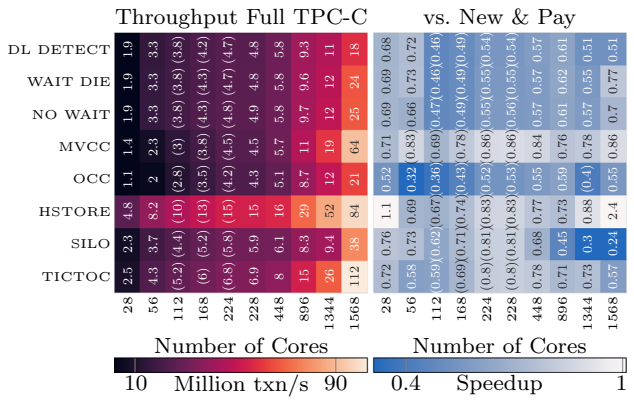


Fig. 22 Detailed throughput for the full TPC-C transaction mix under low conflict on HPE with comparison to the narrow mix. The comparison indicates the speedup ratio by which the throughput differs from the narrow transaction mix (cf. Fig. 13) for a CC scheme (on y-axis) at the same number of cores (on x-axis)

of utilised cores, making for varying relative performance of the CC schemes across the hardware platforms and workload types.

4.3.2 Full TPC-C under low conflict

The previous experiment with the full TPC-C transaction mix under high conflict indicated that besides more conflicts also the higher load on the hardware impacts performance. That is, even for the high conflict workload that generally limits hardware utilisation, the increased load of the heavy transactions influences the CC schemes. Consequently, in the absence of conflicts (low conflict workload), hardware utilisation is the main factor determining the performance of the CC schemes.

First, we provide an overview of the performance of the CC schemes for the full mix on all three hardware platforms (indicating significant differences between those). In the next step, we then contrast the behaviour of the CC schemes for the full mix with the narrow mix on the same hardware platform (i.e. HPE) to identify divergences due to workload characteristics. In the final step, we then compare these divergences of the CC schemes for the full transactions mix across the three hardware platforms to distinguish trends relating to either workload or hardware characteristics.

Comparison of CC Schemes Figure 21 provides an overview of the throughput of all CC schemes for the full TPC-C mix under low conflict on all three hardware platforms. As expected, it shows that the CC schemes have a generally positive scaling behaviour on HPE and Power8, i.e. throughput increases with increasing number of cores. However, in comparison with the narrow transaction mix, throughput is overall lower (cf. Fig. 13). We will compare the narrow mix in detail below.

On Power9, though, the full mix causes anomalous behaviour for all CC schemes, due to a combination of caching and NUMA effects caused by the higher memory footprint. Specifically, pointers to access indexes drop out of the individual processor caches and have to be fetched from potentially distant memory, causing significant slowdown as number of cores increases and subsequently the NUMA distance between them.

We thus further optimised DBx1000 for Power9 by copying these crucial pointers into the local memory of each processor to reduce the memory access cost but have the cores of each processor share these pointers (at most one copy in each processor cache). The results of this optimised variant of Power9 (called *Power9 (RI)*) indeed show a similar behaviour to Power8 and HPE. Notably, this optimisation for Power9 has only minimal effect for the narrow transaction mix, due to the smaller footprint of the involved transactions.

Full vs. Narrow Mix on HPE: In the following, we compare the full and the narrow mix on HPE. On HPE, the transactions of the full TPC-C mix indeed show the expected benefit of MVCC, generally handling read-heavy transactions better. Also under low conflict MVCC becomes third best for the full mix at high core counts, which is different from the narrow mix. Conversely, SILO falls behind for this full mix.

In more detail, Fig. 22 shows the detailed throughput for the full TPC-C mix on HPE and a comparison with the narrow mix. The full mix reduces the throughput of all CC schemes, but distinctly for the individual CC schemes. The best performing HSTORE and the improved MVCC slow down least (HSTORE: 0.69 - 0.88x, MVCC: 0.71 - 0.86x). At highest core count, HSTORE even speeds up by 2.4x and does not thrash as in the narrow mix. TICTOC follows with a

slightly stronger slowdown, especially under NUMA effects at 56 cores and the highest core count. Next, the group of pessimistic locking schemes increasingly slows down until 1344 cores, even more so SILO with a significant slowdown of 0.24x at 1568 cores. Lastly, OCC is significantly affected by the full transaction mix (0.5x). A comparison of the time breakdowns reveals higher coordination costs as the main reason for the overall lower performance for the full transaction mix, i.e. even in this low conflict workload, the heavy transactions increase the time spent for coordination for all CC schemes.

Insight As a major observation, the more involved (i.e. long-running) transactions of the full TPC-C mix do not simply increase the amount of actual work, but their increased footprint indeed impacts concurrency control, for both the high conflict and the low conflict workload. Besides a dampening effect on conflicts and the benefit of MVCC, the individual CC schemes slowdown distinctly. Hence, we conclude that the (read-)heavy transactions of the full TPC-C directly amplify the cost of the individual CC schemes.

Power vs. HPE: Finally, the comparison of the results for the full TPC-C under low conflict across the different hardware platforms confirms the general slowdown on the Power platforms and similar the slowdown trends of most CC schemes[5].

Importantly, the comparison across the hardware platforms confirms our observations on the relation of their hardware characteristics to the behaviour of the CC schemes, albeit leading to different performance. On one hand, the heavier transactions of the full mix cause stronger resource contention on Power, such that on both Power platforms the slowdown at low core counts is stronger than on HPE. On the other hand, the CC schemes scale better on Power, due to their better NUMA characteristics, finally reaching similar or less slowdown than on HPE at highest core counts. Notably, on HPE we previously observed a significant benefit of MVCC handling the read-heavy transactions of the full mix. On Power, the resource contention cancels out this benefit of MVCC.

Insight The full TPC-C transaction mix makes concurrency control even more costly on Power regardless the amount of conflicts in the workload, i.e. larger processor resources on HPE prove more beneficial than the better NUMA characteristics on Power for the full mix in contrast to the narrow mix. Overall, considering both transactions mixes, the CC schemes compare for low conflict workload as follows: (1) HSTORE provides the best performance (on any hardware) as long as there are barely any conflicts, i.e. even few conflicts inhibit its performance (e.g. even low conflict TPC-C at high core counts). (2) TICTOC performs most reliably (even for both low and high conflict workloads). The remaining CC schemes compare diversely. Their performance depends on the characteristics of the individual hardware platforms (NUMA and cache capacity) and the workload. For example, due to the large memory footprint of the read-heavy transactions, MVCC does not prove advantageous on all hardware (i.e. on Power), despite targeting read-heavy transactions.

[5] Detailed figures omitted for brevity are available online [5].

5 Conclusion and future work

In this paper, we presented the results of our extensive analysis of concurrency control on real(ly) large multi-socket hardware as major component of OLTP DBMS. To conclude, we first summarise our major findings. Based on these findings, we then discuss our recommendations as well as possible future directions towards high and robust performing OLTP DBMSs.

5.1 Discussion of findings

In the following, we summarise the main findings of the two evaluation parts and conclude with final insights.

5.1.1 Findings of part one

In the first part of our evaluation, we revisited the simulation of OLTP on then predicted large many-core hardware [71] and compared it to an Intel-based hardware which does provide "a 1000 cores" today but as multi-socket hardware. We identified several discrepancies between the original simulation and real hardware in our evaluation. Importantly, we showed that all CC schemes indeed scale well beyond 1000 cores for low conflict workload, when using state-of-the-art optimisations. Notably, due to shifting bottlenecks, combinations of optimisations were necessary, e.g. hardware-assisted timestamp allocation only improved performance with additional optimisations, as shifting contention then caused latch thrashing. Among the evaluated CC schemes, the now included TICTOC outperformed all other schemes for both low conflict and high conflict workloads. However, all of them including TICTOC still become overwhelmed by conflicts under high contention, degrading even more drastically on our real Intel-based hardware than in the simulation.

5.1.2 Findings of part two

In the second part of our evaluation, we then presented the results of our broadened evaluation, additionally comprising two IBM Power-based hardware platforms and the full transaction mix of TPC-C. With this broadened evaluation, we indeed confirmed our initial observations and further connected the performance of CC schemes with hardware

and workload characteristics, e.g. NUMA effects, processor resources, conflicts, and transaction footprint.

We observed common outstanding and complex nuanced effects of hardware and workload characteristics. First, under high contention, the previously observed thrashing caused by overwhelming conflicts persisted regardless of hardware or other workload characteristics, impeding adequate utilisation of all our large hardware. A major cause of this thrashing of all CC schemes is the simple inter-transaction parallel execution scheme commonly used in today's DBMS [16,32,34,36,51], as it can only utilise high hardware parallelism with high transaction concurrency, amplifying contention. Second, we observed more nuanced effects from the interaction between CC schemes, hardware, and workload. Different hardware characteristics proved significant depending on the design of the CC schemes, e.g. temporary copies of optimistic CC demand cache capacity and bandwidth, while locking of pessimistic CC is latency sensitive. Notably, we observed negative effects only when this demand exceeded the available cache capacity or bandwidth. That is, these capacity related effects did not appear as long as sufficient resources were available. Moreover, the workload further influenced this interaction between the CC schemes and the hardware. For example, the transaction footprint (accessed tuples) amplified the cache demand, degrading performance of optimistic CC when the cache was too small. However, at the same time, the transaction footprint also alleviated contention off latches indeed improving performance of pessimistic CC. Consequently, hardware and workload have complex effects on CC schemes and overall bottlenecks in the DBMS.

5.1.3 Insights from findings

The bottom line of our findings is, that an agglomeration of bottlenecks in the system determines the cost of transaction execution and overall system performance. To reason about the scalability of DBMSs, it thus is important to understand how the cost of individual bottlenecks scales and how these bottlenecks interact. In this regard, our evaluation has shown complex effects of workload and hardware as well as complex interaction of bottlenecks adding up, amplifying each other but also dampening or hiding each other. Then, to achieve high performance, these costs must be balanced against the available hardware resources considering the workload at hand. To conclude, we argue that maintaining the ideal balance despite changing workloads and evolving hardware will enable robust DBMS performance.

5.2 Recommendations and peek into the future

Based on our findings, we now discuss our recommendations to achieve robust DBMS performance and point out according research avenues.

5.2.1 Comprehensive contention management

As discussed above, all evaluated CC schemes scale poorly, surrendering to conflicts and contention. Even advanced CC schemes with conflict mitigation mechanisms do not reliably withstand many conflicts, like TICTOC or beyond the evaluated ones CICADA [26,37,45,68,73]. Moreover, besides logical contention of transaction conflicts, also physical contention (e.g. on latches) significantly impacts performance and system components outside of the CC scheme strongly affect contention (logical & physical), especially the simple but common inter-transaction parallel execution scheme.

As a consequence, we recommend broader system-wide contention management far beyond the CC scheme, since system-wide many factors affect contention and there are more options to efficiently manage contention. For example, system-wide contention management (especially across concurrency control, execution scheme, and scheduler) could reduce logical contention with smarter parallel execution rather than with conflict mitigation in CC schemes, thereby reducing contention more efficiently. As such, we envision contention management throughout the entire system to better balance contention shifting across system components, which is a big challenge in achieving robust performance. While there is work in this direction [13,42,58,66,67,70,74], we believe that extending these to our notion of system-wide contention management and extending to stronger awareness of the underlying hardware would greatly benefit DBMS performance. In particular, interesting directions are broader forms of parallel execution besides inter-transaction parallelism and a transaction scheduler, which is aware of system-wide contention and the interaction with hardware like cache competition. A future route along this line would be to choose the appropriate form of parallel execution of transactions, e.g. inter-transaction parallel execution for uncontended workloads or where contention is "beneficial" (i.e. due to resource sharing), intra-transaction parallel execution for contending transactions, and even sequential execution under excessive contention.

We further recommend adaptive concurrency control as part of comprehensive contention management. Adaptive CC is recognised for employing the most suitable CC scheme for a group of transactions or partition of tuples, e.g. CormCC [53] is an outstanding candidate with low overhead cooperation between a host of CC schemes. We believe that adaptive CC should determine the CC schemes as part of our proposed system-wide contention management, as those CC schemes are strongly affected by many factors of the overall system but in turn strongly affect the system. For example, conflict frequency on a tuple (logical contention) is an important feature to determine the CC scheme, which again is strongly affected by transaction scheduling.

5.2.2 Advanced performance models

Other important aspects shown in our evaluation are the complex dependencies of DBMS performance, i.e. the system design, the workload and the underlying hardware interacting and jointly affecting thrashing points. Due to these complex dependencies, we argue for comprehensive (e.g. learned) performance models that can better reflect these complex effects. Current work on performance models facilitating synthesis of data structures [28] and recent progress on learned components [14,24,38,47] spark our confidence in learned performance models [25] deriving complex dependencies beyond the ability of purely analytical models. Such models would then not only help to inform contention management as discussed above but also would open new opportunities, e.g. finding the optimal hardware for a given workload. In the long term, such performance models would further enable quick exploration of system performance without extensive benchmarking and could eventually lead to performance guarantees of DBMSs.

5.2.3 Adaptive system architectures

Our findings point out that optimal system performance requires the system design to ideally balance bottlenecks. However, this balance differs for workloads as well as hardware and changes over time, due to workload fluctuation but also progress of state-of-the-art (e.g. optimisations). Hence, we advocate for adaptive systems and especially adaptive system architectures capable of re-balancing the system design. Beyond the proposed adaptation and synthesis strategies [27,28,40,53], we argue for flexible system-wide adaptation, which exceeds adaptation of individual components and opposes rigid instance optimisation. Towards effective system-wide adaptation, performance prediction and adaptation overhead are significant challenges. As recommended above, performance predictions will benefit from advanced performance models, whereas we consider flexible system architectures and execution models to drive efficiency. Specifically, we envision the decomposition of system designs into fine-grained building blocks which can be efficiently composed at runtime [4] (e.g. by new compilation techniques). This will enable flexible system architectures to broadly transform a system balancing bottlenecks across all system components. Thereby, we envision adaptive system architectures to successfully adjust to changing workloads and shifting hardware balances.

5.2.4 Artefact availability

Finally, another important route is that data of extensive evaluations like this should be made available for the community. We have released all artefacts [5,6] of this evaluation, making these available to the database community for further research. We believe that despite our extensive analysis in this paper the data itself is a valuable source for future research. So, we hope that this source helps researchers and system builders to further dig into details they find interesting and come up with their own conclusions.

Acknowledgements We would like to express our great gratitude to the authors of [71] for providing DBx1000 as open-source making this work possible. Furthermore, we also would like to thank IBM and HPE for providing us extensive access to the hardware used in this paper.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Advanced Micro Devices, Inc.: AMD EPYC™ 7003 SERIES PROCESSORS (2021). https://www.amd.com/system/files/documents/amd-epyc-7003-series-datasheet.pdf
2. Appuswamy, R., Anadiotis, A.C., Porobic, D., Iman, M.K., Ailamaki, A.: Analyzing the impact of system architecture on the scalability of OLTP engines for high-contention workloads. Proc. VLDB Endow. **11**(2), 121–134 (2017). https://doi.org/10.14778/3149193.3149194
3. Bang, T., May, N., Petrov, I., Binnig, C.: The tale of 1000 cores: an evaluation of concurrency control on real(ly) large multi-socket hardware. In: Proceedings of the 16th International Workshop on Data Management on New Hardware, pp. 3:1–3:9. ACM (2020). https://doi.org/10.1145/3399666.3399910
4. Bang, T., May, N., Petrov, I., Binnig, C.: AnyDB: An architecture-less DBMS for any workload. In: 11th Annual Conference on Innovative Data Systems Research (CIDR '21) (2021). http://cidrdb.org/cidr2021/papers/cidr2021_paper10.pdf
5. Bang, T., May, N., Petrov, I., Binnig, C.: The full story of 1000 cores: an examination of concurrency control on real(ly) large multi-socket hardware - measurements, logs, plots (2021). Archived: https://doi.org/10.48328/tudatalib-726, browsable: https://github.com/DataManagementLab/VLDBJ_1000_cores_measurements
6. Bang, T., May, N., Petrov, I., Binnig, C.: The full story of 1000 cores: An examination of concurrency control on real(ly) large multi-socket hardware - source code (2021). Archived: https://doi.org/10.48328/tudatalib-727, browsable: https://github.com/DataManagementLab/DBx1000
7. Bernstein, P.A., Goodman, N.: Concurrency control in distributed database systems. ACM Comput. Surv. **13**(2), 185–221 (1981)
8. Bernstein, P.A., Goodman, N.: Multiversion concurrency control-theory and algorithms. ACM Trans. Database Syst. **8**(4), 465–483 (1983)
9. Brown, T., Kogan, A., Lev, Y., Luchangco, V.: Investigating the performance of hardware transactions on a multi-socket machine. In: Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, pp. 121–132. ACM, 2935796 (2016). https://doi.org/10.1145/2935764.2935796
10. Celtruda, J.O., Crosthwait, W.R., Earle, J.G., Henderson, R.F., Fennel, J.W.J.: Apparatus and method for serializing instructions from two independent instruction streams (1972). https://worldwide.espacenet.com/patent/search?q=pn%3DCA954227A
11. Clements, A.T., Kaashoek, M.F., Zeldovich, N.: Scalable address spaces using RCU balanced trees. In: Proceedings of the Sev-

enteenth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 199–210. Association for Computing Machinery (2012). https://doi.org/10.1145/2150976.2150998
12. David, T., Guerraoui, R., Trigonakis, V.: Everything you always wanted to know about synchronization but were afraid to ask. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, pp. 33–48. ACM, 2522714 (2013). https://doi.org/10.1145/2517349.2522714
13. Dice, D., Kogan, A.: Avoiding Scalability Collapse by Restricting Concurrency. Lecture Notes in Computer Science, pp. 363–376. Springer International Publishing (2019). https://doi.org/10.1007/978-3-030-29400-7_26
14. Ding, J., Minhas, U.F., Yu, J., Wang, C., Do, J., Li, Y., Zhang, H., Chandramouli, B., Gehrke, J., Kossmann, D., Lomet, D., Kraska, T.: Alex: an updatable adaptive learned index. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, SIGMOD '20, SIGMOD '20, pp. 969–984. ACM (2020). https://doi.org/10.1145/3318464.3389711
15. Drepper, U.: What every programmer should know about memory (2007). https://people.freebsd.org/~lstewart/articles/cpumemory.pdf
16. Färber, F., Cha, S.K., Primsch, J., Bornhövd, C., Sigg, S., Lehner, W.: SAP HANA database: data management for modern business applications. SIGMOD Rec. **40**(4), 45–51 (2012). https://doi.org/10.1145/2094114.2094126
17. Farshin, A., Roozbeh, A., Maguire, G.Q., Kostić, D.: Make the most out of last level cache in Intel processors. In: Proceedings of the Fourteenth EuroSys Conference 2019. Association for Computing Machinery, p. Article 8 (2019). https://doi.org/10.1145/3302424.3303977
18. Franke, H., Russell, R., Kirkwood, M.: Fuss, futexes and furwocks: fast userlevel locking in Linux. In: AUUG Conference Proceedings, vol. 85, pp. 479 – 495. AUUG, Inc. Kensington, NSW, Australia (2002). https://www.kernel.org/doc/mirror/ols2002.pdf#page=479
19. Harding, R., Van Aken, D., Pavlo, A., Stonebraker, M.: An evaluation of distributed concurrency control. Proc. VLDB Endow. **10**(5), 553–564 (2017). https://doi.org/10.14778/3055540.3055548
20. Hennessy, J.L.: Computer architecture: a quantitative approach, 5th edn. Morgan Kaufmann Publication, Burlington (2012)
21. Hewlett Packard Enterprise: The Unique Modular Architecture of HPE Superdome Flex: How it Works and Why it Matters (2018). https://community.hpe.com/t5/Servers-The-Right-Compute/The-unique-modular-architecture-of-HPE-Superdome-Flex-How-it/ba-p/7001330#.XnsMbEBFyAg
22. Hewlett Packard Enterprise Development LP: HPE Superdome Flex, Intel Processors Scale SAP HANA (2018). https://www.intel.com/content/www/us/en/big-data/hpe-superdome-flex-sap-hana-wp.html
23. Hewlett Packard Enterprise Development LP: HPE Superdome Flex Server Architecture and RAS (2020). https://assets.ext.hpe.com/is/content/hpedam/documents/a00036000-6999/a00036491/a00036491enw.pdf
24. Hilprecht, B., Binnig, C., Röhm, U.: Learning a partitioning advisor for cloud databases. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, SIGMOD '20, pp. 143–157. ACM (2020). https://doi.org/10.1145/3318464.3389704
25. Hilprecht, B., Binnig, C.: One model to rule them all: towards zero-shot learning for databases. In: 11th Annual Conference on Innovative Data Systems Research (CIDR '22) (2022)
26. Huang, Y., Qian, W., Kohler, E., Liskov, B., Shrira, L.: Opportunities for optimism in contended main-memory multicore transactions. Proc. VLDB Endow. **13**(5), 629–642 (2020). https://doi.org/10.14778/3377369.3377373
27. Idreos, S., Dayan, N., Qin, W., Akmanalp, M., Hilgard, S., Ross, A., Lennon, J., Jain, V., Gupta, H., Li, D., Zhu, Z.: Learning key-value store design. CoRR (2019). arxiv:1907.05443
28. Idreos, S., Dayan, N., Qin, W., Akmanalp, M., Hilgard, S., Ross, A., Lennon, J., Jain, V., Gupta, H., Li, D., et al.: Design continuums and the path toward self-designing key-value stores that know and learn. In: 9th Annual Conference on Innovative Data Systems Research (CIDR '19) (2019)
29. Intel Corporation: Intel®64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4 (2019)
30. International Business Machines Corporation: POWER8 Processor User's Manual for the Single-Chip Module (Version 1.3) (2016). https://openpowerfoundation.org/?resource_lib=power8-processor-users-manual
31. International Business Machines Corporation: POWER9 Processor User's Manual (Version 2.1) (2019). https://openpowerfoundation.org/?resource_lib=power9-processor-users-manual
32. Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S., Jones, E.P.C., Madden, S., Stonebraker, M., Zhang, Y., Hugg, J., Abadi, D.J.: H-Store: a high-performance, distributed main memory transaction processing system. Proc. VLDB Endow. **1**(2), 1496–1499 (2008). https://doi.org/10.14778/1454159.1454211
33. Kersten, T., Leis, V., Kemper, A., Neumann, T., Pavlo, A., Boncz, P.: Everything you always wanted to know about compiled and vectorized queries but were afraid to ask. Proc. VLDB Endow. **11**(13), 2209–2222 (2018). https://doi.org/10.14778/3275366.3284966
34. Kimura, H.: FOEDUS: OLTP engine for a thousand cores and NVRAM. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 691–706. ACM (2015). https://doi.org/10.1145/2723372.2746480
35. Kung, H.T., Robinson, J.T.: On optimistic methods for concurrency control. ACM Trans. Database Syst. **6**(2), 213–226 (1981)
36. Lahiri, T., Kissling, M.: Oracle's In-Memory Database Strategy for OLTP and Analytics (2015). https://www.doag.org/formes/pubfiles/7378967/2015-K-DB-Tirthankar_Lahiri-Oracle_s_In-Memory_Database_Strategy_for_Analytics_and_OLTP-Manuskript.pdf
37. Lim, H., Kaminsky, M., Andersen, D.G.: Cicada: Dependably fast multi-core in-memory transactions. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp. 21–35. ACM (2017). https://doi.org/10.1145/3035918.3064015
38. Lu, J., Chen, Y., Herodotou, H., Babu, S.: Speedup your analytics: automatic parameter tuning for databases and big data systems. Proc. VLDB Endow. **12**(12), 1970–1973 (2019). https://doi.org/10.14778/3352063.3352112
39. Mulnix, D.L.: Intel®Xeon®Processor Scalable Family Technical Overview (2017). https://www.intel.com/content/www/us/en/developer/articles/technical/xeon-processor-scalable-family-technical-overview.html
40. Porobic, D., Liarou, E., Tozun, P., Ailamaki, A.: Atrapos: adaptive transaction processing on hardware islands. In: 2014 IEEE 30th International Conference on Data Engineering, p. 12 (2014). https://doi.org/10.1109/ICDE.2014.6816692
41. Porobic, D., Pandis, I., Branco, M., Tözün, P., Ailamaki, A.: Characterization of the impact of hardware islands on OLTP. VLDB J. **25**(5), 625–650 (2016). https://doi.org/10.1007/s00778-015-0413-2
42. Prasaad, G., Cheung, A., Suciu, D.: Improving high contention OLTP performance via transaction scheduling, [cs] (2018). arXiv:1810.01997
43. Private conversation with Derek Schumacher, Russ Anderson, and Dimitri Sivanich of Hewlett Packard Enterprise (HPE). 2021-11-16
44. Psaroudakis, I., Scheuer, T., May, N., Sellami, A., Ailamaki, A.: Scaling up concurrent main-memory column-store scans: towards adaptive NUMA-aware data and task placement. Proc.

VLDB Endow. **8**(12), 1442–1453 (2015). https://doi.org/10.14778/2824032.2824043
45. Qadah, T.M., Sadoghi, M.: Quecc: A queue-oriented, control-free concurrency architecture. In: Proceedings of the 19th International Middleware Conference, pp. 13–25. ACM (2018). https://doi.org/10.1145/3274808.3274810
46. SAP SE: SAP HANA Hardware and Cloud Measurement Tools (HCMT) (2020). https://help.sap.com/viewer/02bb1e64c2ae4de7a11369f4e70a6394/2.0/en-US
47. Sheng, Y., Tomasic, A., Zhang, T., Pavlo, A.: Scheduling OLTP transactions via learned abort prediction. In: Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM '19. Association for Computing Machinery (2019). https://doi.org/10.1145/3329859.3329871
48. Sirin, U., Tözün, P., Porobic, D., Ailamaki, A.: Micro-architectural analysis of in-memory OLTP. In: Proceedings of the 2016 International Conference on Management of Data, pp. 387–402. ACM (2016). https://doi.org/10.1145/2882903.2882916
49. Starke, W.J., Dodson, J.S., Stuecheli, J., Retter, E., Michael, B.W., Powell, S.J., Marcella, J.A.: IBM POWER9 memory architectures for optimized systems. IBM J. Res. Dev. **62**(4–5), 3:1-3:13 (2018). https://doi.org/10.1147/JRD.2018.2846159
50. Starke, W.J., Stuecheli, J., Daly, D.M., Dodson, J.S., Auernhammer, F., Sagmeister, P.M., Guthrie, G.L., Marino, C.F., Siegel, M., Blaner, B.: The cache and memory subsystems of the IBM POWER8 processor. IBM J. Res. Dev. **59**(1), 3:1-3:13 (2015). https://doi.org/10.1147/JRD.2014.2376131
51. Stonebraker, M., Rowe, L.A.: The design of postgres. In: Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data, SIGMOD '86, pp. 340–355. Association for Computing Machinery, New York, NY, USA (1986). https://doi.org/10.1145/16894.16888
52. Tanabe, T., Hoshino, T., Kawashima, H., Tatebe, O.: An analysis of concurrency control protocols for in-memory databases with CCBench. Proc. VLDB Endow. **13**(13), 3531–3544 (2020). https://doi.org/10.14778/3424573.3424575
53. Tang, D., Elmore, A.J.: Toward coordination-free and reconfigurable mixed concurrency control. In: 2018 USENIX Annual Technical Conference (USENIX ATC 18). USENIX Association (2018). https://www.usenix.org/conference/atc18/presentation/tang
54. Threading Building Blocks (TBB). https://www.threadingbuildingblocks.org/
55. The PostgreSQL Global Development Group: PostgreSQL 14 released! https://www.postgresql.org/about/news/postgresql-14-released-2318/
56. The Transaction Processing Council: TPC-C benchmark (revision 5.9.0) (2007). http://www.tpc.org/tpcc/spec/tpcc_current.pdf
57. Thomas Neumann, M.F.: Umbra: a disk-based system with in-memory performance. In: 10th Annual Conference on Innovative Data Systems Research (CIDR '20) (2020). http://cidrdb.org/cidr2020/papers/p29-neumann-cidr20.pdf
58. Tian, B., Huang, J., Mozafari, B., Schoenebeck, G.: Contention-aware lock scheduling for transactional databases. Proc. VLDB Endow. **11**(5), 648–662 (2018). https://doi.org/10.1145/3187009.3177740
59. Travis, M.: x86/platform/uv: Add check of tsc state set by uv bios (2017). https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=97d21003df3e7504c899b1701546f18ff475966f
60. Travis, M.: x86/tsc: Add option that tsc on socket 0 being non-zero is valid (2017). https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=341102c3ef29c33611586072363cf9982a8bdb77
61. Travis, M.: x86/tsc: Provide a means to disable tsc art (2017). https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=6c66350d0a482892793b888b07c1177fc6d4b344
62. Tu, S., Zheng, W., Kohler, E., Liskov, B., Madden, S.: Speedy transactions in multicore in-memory databases. In: ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013, pp. 18–32. ACM (2013). https://doi.org/10.1145/2517349.2522713
63. Vetter, S., Caldeira, A.B., Cho, Y., Cruickshank, J., Grabowski, B., Haug, V., Laidlaw, A., Sung, S.Y.: IBM Power Systems E870 and E880 Technical Overview and Introduction (2017). http://www.redbooks.ibm.com/abstracts/redp5137.html?Open
64. Vetter, S., Cruickshank, J., Haug, V., Li (Victor), Y., Röll, A.: IBM Power Systems E980: Technical Overview and Introduction (2020). http://www.redbooks.ibm.com/abstracts/redp5510.html?Open
65. Viswanathan, V., Kumar, K., Willhalm, T., Lu, P., Filipiak, B., Sakthivelu, S.: Intel memory latency checker v3.8 (2020). https://software.intel.com/en-us/articles/intelr-memory-latency-checker
66. Wang, D., Cai, P., Qian, W., Zhou, A.: Discriminative admission control for shared-everything database under mixed OLTP workloads. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE), pp. 780–791. IEEE (2021). https://doi.org/10.1109/ICDE51399.2021.00073
67. Wang, J., Guo, J., Zhou, H., Cai, P., Qian, W.: Adaptive transaction scheduling for highly contended workloads. In: Li, G., Yang, J., Gama, J., Natwichai, J., Tong, Y. (eds.) Database Systems for Advanced Applications, pp. 576–580. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-18590-9_90
68. Wang, T., Kimura, H.: Mostly-optimistic concurrency control for highly contended dynamic workloads on a thousand cores. Proc. VLDB Endow. **10**(2), 49–60 (2016). https://doi.org/10.14778/3015274.3015276
69. Wu, Y., Arulraj, J., Lin, J., Xian, R., Pavlo, A.: An empirical evaluation of in-memory multi-version concurrency control. Proc. VLDB Endow. **10**(7), 781–792 (2017). https://doi.org/10.14778/3067421.3067427
70. Yi, Z., Yao, Y., Chen, K.: Ftsd: a fissionable lock for multicores. In: Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems, pp. 123–130. ACM (2021). https://doi.org/10.1145/3476886.3477518
71. Yu, X., Bezerra, G., Pavlo, A., Devadas, S., Stonebraker, M.: Staring into the abyss: an evaluation of concurrency control with one thousand cores. Proc. VLDB Endow. **8**(3), 209–220 (2014). https://doi.org/10.14778/2735508.2735511
72. Yu, X., Bezerra, G., Pavlo, A., Devadas, S., Stonebraker, M.: DBx1000 github (commit: b40c09a) (2020). https://github.com/yxymit/DBx1000/tree/b40c09a27d9ab7a4c2222e0ed0736a0cb67b7040
73. Yu, X., Pavlo, A., Sanchez, D., Devadas, S.: TicToc: Time traveling optimistic concurrency control. In: Proceedings of the 2016 International Conference on Management of Data, pp. 1629–1642. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2882903.2882935
74. Zhang, T., Tomasic, A., Sheng, Y., Pavlo, A.: Performance of OLTP via intelligent scheduling. In: 34th International Conference on Data Engineering (ICDE), EEE Computer Society, Paris, France (2018). https://doi.org/10.1109/ICDE.2018.00132