Performance Characterization of HTAP Workloads

Utku Sirin
EPFL
utku.sirin@epfl.ch

Sandhya Dwarkadas
University of Rochester
sandhya@cs.rochester.edu

Anastasia Ailamaki
EPFL
anastasia.ailamaki@epfl.ch

2021 IEEE 37th International Conference on Data Engineering (ICDE) | 978-1-7281-9184-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICDE51399.2021.00162

***Abstract*—Hybrid Transactional and Analytical Processing (HTAP) systems have become popular in the past decade. HTAP systems allow running transactional and analytical processing workloads on the same data and hardware. As a result, they suffer from workload interference. Despite the large body of existing work in HTAP systems and architectures, none of the existing work has systematically analyzed workload interference for HTAP systems.**

In this work, we characterize workload interference for HTAP systems. We show that the OLTP throughput drops by up to 42% due to sharing the hardware resources. Partitioning the last-level cache (LLC) among the OLTP and OLAP workloads can significantly improve the OLTP throughput without hurting the OLAP throughput. The OLAP throughput is significantly reduced due to sharing the data. The OLAP execution time is exponentially increased if the OLTP workload generates fresh tuples faster than the HTAP system propagates them. Therefore, in order to minimize the workload interference, HTAP systems should isolate the OLTP and OLAP workloads in the shared hardware resources and should allocate enough resources to fresh tuple propagation to propagate the fresh tuples faster than they are generated.

***Index Terms*—Hybrid transactional and analytical processing, hybrid workloads, HTAP, real-time analytics, workload characterization, micro-architectural analysis, modern hardware, hardware-software co-design.**

I. Introduction

Hybrid Transactional and Analytical Processing (HTAP) systems have gained significant attention in the past decade. HTAP systems combine Online Transactional Processing (OLTP) and Online Analytical Processing (OLAP) systems into a single, unified system providing real-time analytical query processing over fresh, transactional data. Numerous applications benefit from real-time analytical query processing such as fraud detection, risk analysis, and IoT applications [1].

There is a significant amount of work on HTAP systems and architectures. [2]–[4] propose two-copy HTAP architectures, where the OLTP and OLAP workloads run on their private copies of the data. [5]–[11] propose single-copy HTAP architectures, where the OLTP and OLAP workloads run on the same copy of the data. The OLTP and OLAP workloads share data and hardware resources both for the two- and single-copy architectures. As a result, they suffer from workload interference in the shared data and hardware resources.

Despite the large body of existing work on HTAP systems, the existing work falls short in systematically analyzing workload interference for HTAP systems. In this work, we fill this gap and highlight a research direction to mitigate workload interference for HTAP systems. We demonstrate the following:

- OLTP and OLAP workloads interfere with each other in the last-level cache and memory bandwidth. While the OLAP throughput drops only by a small amount (by less than 10%), the OLTP throughput drops by a significant amount (by up to 42%). Partitioning the last-level cache among the concurrently running OLTP and OLAP workloads can significantly improve the OLTP throughput without reducing the OLAP throughput. While OLTP always benefits from a larger amount of caches, OLAP does not benefit from more than a few megabytes of caches. Therefore, HTAP systems should isolate the OLTP and OLAP workloads in the shared last-level cache and adopt resource management algorithms based on the needs of the OLTP and OLAP workloads.
- OLTP and OLAP workloads interfere with each other in the shared data. While the OLTP throughput remains the same whether the data is shared or not, the OLAP throughput significantly drops when the data is shared. In particular, the OLAP query execution time is exponentially increased when the OLTP side generates fresh tuples faster than the HTAP system can propagate them from the OLTP to the OLAP side. Therefore, HTAP systems should allocate enough resources to fresh-tuple propagation to make sure that fresh tuples are propagated faster than they are generated to prevent an exponentially increased query execution time.

II. Background and Related Work

HTAP architectures: Three main HTAP architectures have been proposed [1]. The first is two-copy, mixed-format (TCMF) architecture [2]–[4], [12]. TCMF keeps two copies of the data, one for the OLTP side in row format and one for the OLAP component in columnar format. To keep the data consistent across the OLTP and OLAP components, TCMF uses an intermediate data structure, *delta*, that keeps track of the recently modified, fresh tuples. Periodically, TCMF propagates the fresh tuples from the OLTP side to the OLAP side by scanning the delta.

The second is single-copy, mixed-format (SCMF) architecture [5], [6]. SCMF keeps a single copy of the data and uses the intermediate data structure *delta* as the OLTP store. OLTP transactions only modify the delta, whereas the OLAP queries read both the delta and the main copy of the data. The delta is in row format, whereas the main copy of the data is in columnar format.

978-1-7281-9184-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDE51399.2021.00162

Authorized licensed use limited to: UNIVERSITY OF ROCHESTER. Downloaded on September 15,2021 at 21:46:04 UTC from IEEE Xplore. Restrictions apply.

Lastly, single-copy, single-format (SCSF) architecture keeps a single copy of the data and uses a single format, i.e., only row or columnar, both for OLTP and OLAP workloads [7]–[9]. It relies on copy-on-write snapshotting or multi-version concurrency control mechanisms to keep multiple versions of the data, through which analytical queries can access the most recent transactional data.

In all the three architectures, the data is logically shared among the OLTP and OLAP components. When we refer to sharing the data, we refer to sharing the logical data throughout the paper.

Workload characterization: There is a large body of work on database workload characterization. Ailamaki et al. [13] and Hardavellas et al. [14] characterize OLTP and OLAP workloads. Tozun et al. [15], [16] and Sirin et al. [17] dive deep into OLTP workloads and characterize disk-based and in-memory OLTP workloads.

Sirin et al. [18] characterize a wide range of OLAP workloads with a particular focus on column stores. Kersten et al. [19] characterize OLAP workloads with a particular focus on vectorized versus compiled engines. Sompolski et al. [20] also compare vectorized and compiled engines with a focus on predication and SIMD.

The existing work on HTAP systems falls short in analyzing workload interference for HTAP systems. The existing work in database workloads characterization examines either the OLTP or the OLAP workloads. In this work, we take the next step and examine HTAP workloads with a focus on interference.

III. Setup and Methodology

Benchmarks: We use two micro-benchmarks and the CH-benchmark [21]. The micro-benchmarks use the CH-benchmark schema. The first micro-benchmark contains a projection query and an OLTP transaction. The projection query does a SUM() over the *ol_amount* column of the orderline table. The OLTP transaction continuously updates the same *ol_amount* column. The second micro-benchmark contains a join query and the same OLTP transaction as the first micro-benchmark. The join query joins the orderline and orders tables and does a SUM() over the *ol_amount* column. The joins use the hash-join algorithm. We chose a projection- and join-based query as they represent the two main data access patterns for OLAP workloads: sequential-scan and random-access [18].

We use Q3 and Q6 of the CH-benchmark, as they represent the two main categories of analytical queries: (i) non-join and (ii) join-intensive queries [18]. We use the New-Order transaction of the CH-benchmark, as it is the heaviest transaction in the CH-benchmark.

Hardware: We use a commodity Intel server with the Broadwell micro-architecture. Table I presents the server parameters. We use Intel's Memory Latency Checker (MLC) [22] to measure the cache-miss latencies and maximum memory bandwidth.

HTAP systems: We examine (i) HTAP engine of a traditional, commercial system, DBMS A, (ii) a popular, new-generation, commercial HTAP system, DBMS B, (iii) and an academic prototype that we built based on an existing open-source OLTP system, Silo [23], and an existing open-source OLAP system, Typer [19]. We name our system Siper, which combines the initial letters of Silo and final letters of Typer.

TABLE I
Server parameters.

Processor	Intel(R) Xeon(R) CPU E5-2680 v4 (Broadwell)
#sockets	2
#cores per socket	14
Hyper-threading	Off
Turbo-boost	Off
Clock speed	2.40GHz
Per-socket bandwidth	66GB/s (sequential) 60GB/s (random)
L1I / L1D (per core)	32KB / 32KB 16-cycle miss latency
L2 (per core)	256KB 26-cycle miss latency
L3 (shared)	(inclusive) 35MB 160-cycle miss latency
Memory	256GB

DBMS A uses the two-copy, mixed-format (TCMF) HTAP architecture. DBMS B uses the single-copy, mixed-format (SCMF) HTAP architecture. Siper also uses the TCMF architecture. We did not study the single-copy, single-format (SCSF) architecture. However, we believe our conclusions apply to SCSF. SCSF suffers from traversing version-chains, whereas TCMF and SCMF suffer from processing the delta structure. Both challenges are random-accesses-intensive. Both challenges depend on how fast the OLTP side produces fresh tuples and how fast the HTAP system can merge the fresh tuples into the main copy of the data.

OS & Compiler: We use Ubuntu 16.04.6 LTS and gcc 5.4.0.

VTune: We use Intel VTune 2020. We use VTune's built-in memory-access analysis to measure memory bandwidth consumption. As we use a single socket for all of our experiments, we report the average per-socket bandwidth values.

Measurements: We first populate the database and generate statistics. Lastly, we perform a three-minute warmup period, followed by a ten-minute throughput-measurement or VTune-profiling period. We disable hyper-threading and turbo-boost, as they jeopardize VTune counter values [24], except in Section V-B where we examine hyper-thread-sharing.

We use only a single socket for all of our experiments, except in Section VI. We disable the cores in the other socket and use the relevant OS interface to make sure memory is always locally allocated. We do single- and multi-threaded experiments. We choose a scaling factor of 350 (a database of 30GB) for all the experiments.

We disable compression for DBMS A. DBMS B does not expose a parameter to enable/disable the compression. It uses an internally-decided compression scheme whose details are not revealed. Siper does not rely on any compression scheme.

Authorized licensed use limited to: UNIVERSITY OF ROCHESTER. Downloaded on September 15,2021 at 21:46:04 UTC from IEEE Xplore. Restrictions apply.

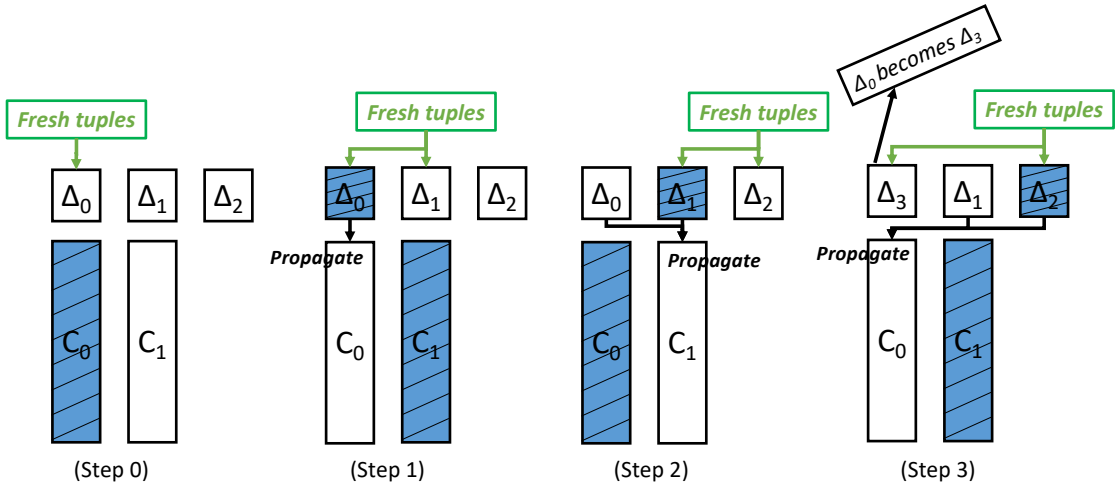

Fig. 1. The OLAP side of Siper. The OLTP side continuously produces fresh tuples. The active column and delta combination at each step is colored by blue and marked by left-striped lines.

IV. Siper

In this section, we present Siper. Siper relies on the TCMF architecture (see Section II). Furthermore, it keeps two copies of the modified columns at the OLAP side. One of the copies is used for propagating the fresh tuples, the other copy is used for query processing. Figure 1 presents how the OLAP side of Siper works.

At step 0, the query is executed on the base column over the first copy of the column, i.e., C_0 (colored by blue and marked by left-stripes). During step 0, the fresh tuples are tracked by Δ_0. At step 1, the query is executed on C_1 and Δ_0, and Δ_0 is propagated to C_0. At step 2, C_0 (which now has Δ_0 propagated) and Δ_1 are used to serve the query, during which Δ_0 and Δ_1 are propagated to C_1. We keep the fresh tuples both in Δ_1 and Δ_2. At step 3, and onwards the system starts iterating itself. At step i, the system concurrently executes the following:

- Processes $\Delta_{(i-1)\%3}$ and $C_{i\%2}$ (for OLAP queries)
- Propagates the $\Delta_{(i-1)\%3}, \Delta_{(i-2)\%3}$
- Keeps track of the fresh tuples by $\Delta_{i\%3}, \Delta_{(i-1)\%3}$

We keep track of the fresh tuples in $\Delta_{i\%3}$ and $\Delta_{(i-1)\%3}$ rather than only in $\Delta_{i\%3}$ in order to be able to alternate between the two column copies while also keeping them within two deltas of each other.

V. Interference In Shared Hardware Resources

In this section, we examine interference in the shared hardware resources. We examine the interference in the last-level cache (LLC), memory bandwidth and hyper-threads by using the micro-benchmark and the CH-benchmark. Our goal is to answer the following questions:

- How much do the OLTP and OLAP throughput drop due to the interference?
- Do the OLTP and OLAP throughput drop by the same level?
- Does the OLAP throughput drop by the same level for all the queries and transactions?
- Does the OLTP throughput drop by the same level for all the queries and transactions?

A. LLC and Memory Bandwidth Interference

In this section, we examine the interference in the LLC and memory bandwidth. We examine three configurations: 1 OLTP thread concurrently running with 13 OLAP threads (1T+13A), 7 OLTP thread with 7 OLAP threads (7T+7A), 13 OLTP thread with 1 OLAP threads (13T+1A). For each configuration, we measure the OLTP and OLAP throughput. Then, we divide this throughput by the OLTP (or OLAP) throughput measured when running the OLTP (or OLAP) alone on the same server with the same number of threads. For example, we measure the OLTP and OLAP throughput for the 1T+13A configuration. Then, we measure the OLTP throughput when running the same OLTP workload alone on the same hardware with 1 OLTP thread. We divide the OLTP throughput for the 1T+13A configuration by the OLTP throughput when running the OLTP alone and report this number as the interference at the OLTP side. We do the same to report the interference at the OLAP side. Workloads are always on a single socket using locally allocated memory. For Siper, we use 1 thread for delta propagation. We use 12 instead of 13 OLAP threads for the 1T+13A configuration.

Table II presents the results. DBMS A and B lightly suffer from the interference. Both the OLTP and OLAP throughput drop by 1-12%. Siper significantly suffers from the interference at the OLTP side. The OLTP throughput drops by 11-42%. The larger the number of OLAP threads is, the more significant the interference at the OLTP side is. Siper lightly suffers from the interference at the OLAP side. The OLAP throughput drops only by 1-5%.

In Table III, we examine the consumed memory bandwidth. We examine the OLTP-alone and OLAP-alone configurations

Authorized licensed use limited to: UNIVERSITY OF ROCHESTER. Downloaded on September 15,2021 at 21:46:04 UTC from IEEE Xplore. Restrictions apply.

TABLE II
INTERFERENCE IN THE LLC AND MEMORY BANDWIDTH. 1T+13A REFERS TO THE CONFIGURATION RUNNING 1 OLTP AND 13 OLAP THREADS. THE CELLS ARE COLOR-CODED. THE DARKER A CELL IS, THE MORE SIGNIFICANT THE INTERFERENCE IS.

				1T+13A	7T+7A	13T+1A
DBMS A	μ-bench.	Proj.	OLTP	0.99	0.98	0.97
			OLAP	0.98	0.99	1.00
		Join	OLTP	0.96	0.99	0.97
			OLAP	0.94	0.92	1.00
	CH-bench.	Q6	OLTP	0.99	0.99	0.96
			OLAP	1.04	1.01	1.00
		Q3	OLTP	1.05	0.88	0.96
			OLAP	1.01	0.94	0.96
DBMS B	μ-bench.	Proj.	OLTP	0.97	0.91	0.97
			OLAP	0.98	0.97	1.00
		Join	OLTP	0.94	0.89	0.98
			OLAP	1.02	0.98	0.95
	CH-bench.	Q6	OLTP	1.01	0.88	0.97
			OLAP	1.04	0.98	1.01
		Q3	OLTP	0.99	0.91	0.98
			OLAP	1.02	0.97	1.01
Siper	μ-bench.	Proj.	OLTP	0.58	0.73	0.80
			OLAP	0.98	0.99	0.95
		Join	OLTP	0.78	0.82	0.89
			OLAP	0.99	0.98	0.98

TABLE III
CONSUMED MEMORY BANDWIDTH IN GB/S. THE CELLS ARE COLOR-CODED. THE DARKER A CELL IS, THE HIGHER THE BANDWIDTH CONSUMPTION IS. THE MAXIMUM AVAILABLE BANDWIDTH IS 66 GB/S.

		1 thread	7 threads	13 threads
DBS A	Update-only xact.	0.05	0.90	2.49
	Projection	0	1.90	3.37
	Join	0.09	2.80	3.30
DBS B	Update-only xact.	0.02	0.23	0.47
	Projection	0.04	5	7.18
	Join	0.07	2.40	4.36
Siper	Update-only xact.	0.10	4.10	8.30
	Projection	5.4	31.65	58.55
	Join	1.19	12.10	21.71

for varying number of threads. We observe that the larger the amount of bandwidth the OLAP (or OLTP) side of a system consumes, the more the OLTP (or OLAP) side of the system suffers from the interference. Both the OLTP and OLAP sides of DBMS A and B consume a small amount of memory bandwidth (0.02 to 7.18 GB/s), and both the OLTP and OLAP sides of DBMS A and B lightly suffer from the interference. The OLTP side of Siper consumes a small amount of bandwidth (0.10 to 8.30 GB/s), and the OLAP side of Siper lightly suffers from the interference. The OLAP side of Siper consumes a significant amount of bandwidth (5.40 to 58.55 GB/s), and the OLTP side of Siper significantly suffers from the interference.

LLC vs. memory bandwidth: We further study how much of the interference is due to LLC and how much of the interference is due to memory bandwidth for Siper. We use Intel's Cache Allocation Technology (CAT) to isolate LLC among the OLTP and OLAP workloads [25], [26]. For 1 OLTP and 12 OLAP threads, we increase the amount of LLC allocated to OLTP from 1.75MB to 33.25MB. We reserve the remainder of the LLC for OLAP. We run the fresh tuple propagation thread pinned to a separate physical core.

TABLE IV
LLC PARTITIONING EXPERIMENTS FOR SIPER USING 1 OLTP, 1 DELTA PROPAGATION AND 12 OLAP THREADS. THE CELLS ARE COLOR-CODED. THE DARKER A CELL IS, THE MORE SIGNIFICANT THE INTERFERENCE IS.

		LLC partition size for OLTP in MBs				
		1.75	8.75	17.5	31.5	33.25
Projection	OLTP	0.68	0.73	0.77	0.83	0.86
	OLAP	1.00	1.00	1.00	0.98	0.47
Join	OLTP	0.77	0.81	0.83	0.86	0.84
	OLAP	1.00	0.99	0.98	0.90	0.68

Table IV presents the OLTP and OLAP throughput normalized to running on the server alone. The table shows that the projection throughput does not increase for more than 3.5MB LLC. Hence, the sequential-scan-heavy query requires only a certain amount of LLC, after which the increased cache partition size does not increase the throughput. A similar conclusion applies to the join query. More than 3.5MB of LLC improves join query throughput by only 10% more.

When LLC is not partitioned, the OLTP throughput drops by 42% when running with the projection query (see Table II). Allocating 1.75MB of LLC to the OLTP and the rest to the OLAP, the OLTP throughput drops by 32%. Hence, partitioning LLC among OLTP and OLAP is always beneficial for OLTP when running with the projection query, even though the allocated LLC size is minimal. Having increased the OLTP's LLC partition size to 33.25MB, the OLTP's throughput-drop is reduced to 14%. We can surmise that out of the 42% throughput-drop due to the interference in the shared LLC and memory bandwidth, 14% is likely due to contention for memory bandwidth, whereas, $42 - 14 = 28\%$ is due to interference in the LLC.

The interference in the LLC and memory bandwidth is 22% for OLTP when running with the join query. By splitting LLC equally among the OLTP and OLAP workloads (17.5MB to each), the interference at the OLTP side is reduced to 17% without hurting the OLAP performance. If we allocate 33.25MB to OLTP, the interference is further reduced to $\sim$15%. This shows that the 15% interference out of the 22% is likely due to contention for memory bandwidth, with $22 - 15 = 7\%$ due to interference in the LLC.

Based on our evaluation, we conclude that LLC partitioning can be significantly useful to isolate the LLC accesses of sequential-scan-heavy OLAP queries from that of the OLTP transactions. Thanks to the LLC partitioning, the interference at the OLTP side can be reduced from 42% to 14%, without hurting the OLAP performance.

B. Hyper-threads

In this section, we examine the interference in the shared hyper-thread resources. We run 2 OLTP threads on two separate physical cores. We then add 2 OLAP threads such that each OLTP thread shares its core with an OLAP thread, and we examine how much the newly added OLAP threads decrease the throughput of the already running OLTP threads. We do the same by starting with 2 OLAP threads, adding 2 OLTP

Authorized licensed use limited to: UNIVERSITY OF ROCHESTER. Downloaded on September 15,2021 at 21:46:04 UTC from IEEE Xplore. Restrictions apply.

TABLE V

INTERFERENCE IN THE SHARED HYPER-THREADS. 2T+2A REFERS TO THE CONFIGURATION RUNNING 2 OLTP AND 2 OLAP THREADS. THERE IS ALWAYS 1 DELTA PROPAGATION THREAD. THE CELLS ARE COLOR-CODED. THE DARKER A CELL IS, THE MORE SIGNIFICANT THE INTERFERENCE IS.

		2T+2A
Projection	OLTP	0.73
	OLAP	0.55
Join	OLTP	0.77
	OLAP	0.69

TABLE VI

INTERFERENCE IN THE SHARED DATA AT THE OLAP SIDE. THERE IS ALWAYS 1 DELTA PROPAGATION THREAD. NORMALIZED QUERY EXECUTION TIMES. THE CELLS ARE COLOR-CODED. THE DARKER A CELL IS, THE MORE SIGNIFICANT THE INTERFERENCE IS.

		Number of OLTP threads				
		1	7	14	21	28
Number of OLAP threads	1	1.0	1.2	4.2	11.2	19.6
	10	1.0	1.1	4.0	8.9	13.9
	26	1.0	1.1	2.4	4.9	10.8

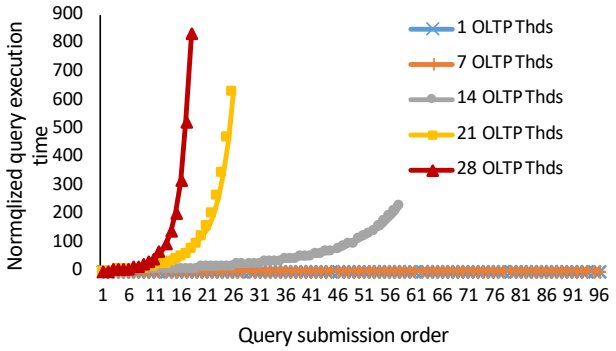

Fig. 2. Interference in the shared data for 10 OLAP threads and 1 delta propagation thread. Normalized query execution times.

threads, and measuring how much the OLAP throughput is decreased. Table V presents the results.

We first examine the projection query. The OLAP threads cause 27% throughput-drop for the OLTP threads, whereas the OLTP threads cause 45% throughput-drop for the OLAP threads. This shows that the sequential-scan-heavy projection query is sensitive to hyper-thread sharing. The speed of sequential data-scan depends on successfully streaming the read requests. If the stream is intervened, e.g., due to a micro-architectural resource being occupied by the sibling hyper-thread, its speed is significantly decreased.

The join query is modestly affected by the sibling OLTP threads. Similarly, the OLTP threads are modestly affected by the sibling join threads. These show that the random-access-heavy OLTP-workload and join-query are more robust to hyper-thread sharing.

VI. INTERFERENCE IN SHARED DATA

In this section, we examine interference in the shared data. We only use Siper, as we do not have control over core and data affinity for DBMS A and B. We place the OLTP and OLAP threads and data in two separate sockets, so that OLTP and OLAP do not share any hardware resources. We place the delta propagation threads and the delta itself in the OLAP socket. We use the projection query and the update-only transaction. Our goal is to answer the following questions:

- How much do the OLTP and OLAP throughput drop due to the interference?
- Do the OLTP and OLAP throughput drop by the same level?
- Does the OLAP throughput drop by the same level for different number of OLTP and OLAP threads?
- Does the OLTP throughput drop by the same level for different number of OLTP and OLAP threads?

First, we examine the interference at the OLTP side. The OLTP throughput is not affected by the interference. The OLTP component we use, i.e., Silo, relies on multi-versioning to keep consistent snapshots of the data. Therefore, the readers do not block the writers, and the OLTP throughput is not affected by the OLAP workload.

Next, we examine the interference at the OLAP side. We use 1 fresh-tuple-propagation thread and examine how much the query execution time is increased at the OLAP side. Table VI presents the results for the projection query and the update-only transaction. The query execution time is increased by a small amount for 1 and 7 OLTP threads and by a large amount for 14 and more OLTP threads.

For 10 OLAP threads and 1 fresh-tuple-propagation thread, we examine the individual query execution times. We submit the queries one-after-the-other. We measure the execution time of each query. Then, we normalize the query execution times of the series of submitted queries based on the execution time of the first submitted query. We plot the normalized query execution time numbers in Figure 2 with increasing numbers of OLTP threads. The query execution time is stable for 1 and 7 OLTP threads, whereas it is exponentially increased for 14 and more OLTP threads.

We examine two parameters to understand the behavior. We examine: (i) fresh tuple generation throughput, which we call λ, and (ii) fresh tuple propagation throughput, which we call μ. λ is approximately the OLTP throughput in our micro-benchmark, as the OLTP transaction we use generates 1 fresh tuple per transaction.

We examine λ/μ. The λ/μ metric represents how fast the fresh tuples are generated compared to how fast the fresh tuples can be propagated. If the λ/μ value is less than 1, it means the OLTP side generates fresh tuples slower than the fresh tuple propagation thread propagates the fresh tuples. If the λ/μ value is greater than 1, it means the OLTP side generates fresh tuples faster than the fresh tuple propagation thread propagates the fresh tuples.

Table VII presents the λ/μ values for 10 OLAP, 1 fresh tuple propagation, and increasing number of OLTP threads. The OLTP component generates fresh tuples slower than they are propagated for 1 and 7 OLTP threads, whereas the OLTP

Authorized licensed use limited to: UNIVERSITY OF ROCHESTER. Downloaded on September 15,2021 at 21:46:04 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
FRESH TUPLE GENERATION THROUGHPUT (λ) VS. FRESH TUPLE PROPAGATION THROUGHPUT (μ).

	Number of OLTP threads				
	1	7	14	21	28
λ/μ	0.1	0.62	0.96	1.26	1.7

side generates fresh tuples almost as fast as or even faster than the fresh tuples are propagated for 14 and more OLTP threads. Therefore, the query execution time is exponentially increased if the OLTP component generates the fresh tuples as fast as or faster than the fresh tuple propagation thread propagates them. HTAP systems should allocate enough resources to fresh tuple propagation such that the fresh tuples can be propagated faster than they are generated by the OLTP component.

VII. CONCLUSIONS

HTAP systems combine OLTP and OLAP workloads into a single workload, where the OLTP and OLAP workloads share both data and hardware resources. In this work, we characterize workload interference for HTAP workloads. We examine the interference both in the shared hardware resources and in the shared data.

We show that interference in the last-level cache (LLC) and contention for memory bandwidth can reduce OLTP throughput by up to 42%. Partitioning the LLC among the OLTP and OLAP workloads can significantly improve OLTP throughput. The interference in the shared data results in an exponentially increased OLAP query execution time if the OLTP workload generates fresh tuples faster than the HTAP system can propagate them. Therefore, in order to minimize workload interference for HTAP systems, HTAP systems should isolate the OLTP and OLAP workloads in their shared-resource accesses and should adopt resource management algorithms that allow fresh tuple propagation at a rate faster than they are generated.

VIII. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers and the member of DIAS laboratory for their feedback. This project has received funding from the US National Science Foundation (NSF) Award CNS1900803, the European Union Seventh Framework Programme (ERC-2013-CoG), under grant agreement no 617508 (ViDa), and Swiss National Science Foundation, Project No.: 200021 146407/1 (Workload- and hardware-aware transaction processing).

REFERENCES

[1] F. Özcan, Y. Tian, and P. Tözün, "Hybrid Transactional/Analytical Processing: A Survey," in *SIGMOD*, 2017, pp. 1771–1775.

[2] P. Larson, A. Birka, E. N. Hanson, W. Huang, M. Nowakiewicz, and V. Papadimos, "Real-Time Analytical Processing with SQL Server," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1740–1751, 2015.

[3] T. Lahiri, S. Chavan, M. Colgan, D. Das, A. Ganesh, M. Gleeson, S. Hase, A. Holloway, J. Kamp, T. Lee, J. Loaiza, N. Macnaughton, V. Marwah, N. Mukherjee, A. Mullick, S. Muthulingam, V. Raja, M. Roth, E. Soylemez, and M. Zait, "Oracle Database In-Memory: A Dual Format In-memory Database," in *ICDE*, 2015, pp. 1253–1258.

[4] D. Makreshanski, J. Giceva, C. Barthels, and G. Alonso, "BatchDB: Efficient Isolated Execution of Hybrid OLTP+OLAP Workloads for Interactive Applications," in *SIGMOD*, 2017, pp. 37–50.

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

[6] A. Skidanov, A. J. Papito, and A. Prout, "A Column Store Engine for Real-time Streaming Analytics," in *ICDE*, 2016, pp. 1287–1297.

[7] T. Neumann, T. Mühlbauer, and A. Kemper, "Fast serializable multi-version concurrency control for main-memory database systems," in *SIGMOD*, 2015, pp. 677–689.

[8] R. Appuswamy, M. Karpathiotakis, D. Porobic, and A. Ailamaki, "The Case For Heterogeneous HTAP," in *CIDR*, 2017.

[9] A. Raza, P. Chrysogelos, A. G. Anadiotis, and A. Ailamaki, "Adaptive HTAP through Elastic Resource Scheduling," in *SIGMOD*, 2020, pp. 2043–2054.

[10] M. Athanassoulis, K. S. Bøgh, and S. Idreos, "Optimal Column Layout for Hybrid Workloads," *Proc. VLDB Endow.*, vol. 12, no. 13, pp. 2393–2407, 2019.

[11] J. Arulraj, A. Pavlo, and P. Menon, "Bridging the Archipelago between Row-Stores and Column-Stores for Hybrid Workloads," in *SIGMOD*, 2016, pp. 583–598.

[12] L. Li, G. Wu, G. Wang, and Y. Yuan, "Accelerating Hybrid Transactional/Analytical Processing Using Consistent Dual-Snapshot," in *Database Systems for Advanced Applications*, 2019, pp. 52–69.

[13] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, "DBMSs on a Modern Processor: Where Does Time Go?" in *VLDB*, 1999, pp. 266–277.

[14] N. Hardavellas, I. Pandis, R. Johnson, N. Mancheril, A. Ailamaki, and B. Falsafi, "Database Servers on Chip Multiprocessors: Limitations and Opportunities," in *CIDR*, 2007, pp. 79–87.

[15] P. Tözün, B. Gold, and A. Ailamaki, "OLTP in Wonderland: Where Do Cache Misses Come From in Major OLTP Components?" in *Damon*, 2013, p. 8.

[16] P. Tözün, I. Pandis, C. Kaynak, D. Jevdjic, and A. Ailamaki, "From A to E: Analyzing TPC's OLTP Benchmarks: The Obsolete, The Ubiquitous, The Unexplored," in *EDBT*, 2013, pp. 17–28.

[17] U. Sirin, P. Tözün, D. Porobic, and A. Ailamaki, "Micro-architectural Analysis of In-memory OLTP," in *SIGMOD*, 2016, pp. 387–402.

[18] U. Sirin and A. Ailamaki, "Micro-architectural Analysis of OLAP: Limitations and Opportunities," *Proc. VLDB Endow.*, vol. 13, no. 6, pp. 840–853, 2020.

[19] T. Kersten, V. Leis, A. Kemper, T. Neumann, A. Pavlo, and P. Boncz, "Everything You Always Wanted to Know About Compiled and Vectorized Queries but Were Afraid to Ask," *Proc. VLDB Endow.*, vol. 11, no. 13, pp. 2209–2222, 2018.

[20] J. Sompolski, M. Zukowski, and P. A. Boncz, "Vectorization vs. Compilation in Query Execution," in *Damon*, 2011, pp. 33–40.

[21] R. Cole, F. Funke, L. Giakoumakis, W. Guy, A. Kemper, S. Krompass, H. Kuno, R. Nambiar, T. Neumann, M. Poess, K.-U. Sattler, M. Seibold, E. Simon, and F. Waas, "The Mixed Workload CH-BenCHmark," in *DBTest*, 2011.

[22] Intel, "Intel Memory Latency Checker," 2020, https://software.intel.com/en-us/articles/intelr-memory-latency-checker.

[23] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden, "Speedy Transactions in Multicore In-memory Databases," in *SOSP*, 2013, pp. 18–32.

[24] Intel, "Understanding How General Exploration Works in Intel VTune Amplifier," 2018, https://software.intel.com/en-us/articles/understanding-how-general-exploration-works-in-intel-vtune-amplifier-xe.

[25] ——, "Introduction to Intel Cache Allocation Technology." 2016, https://software.intel.com/content/www/us/en/develop/articles/introduction-to-cache-allocation-technology.html.

[26] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *ISCA*, 2015, pp. 450–462.

Authorized licensed use limited to: UNIVERSITY OF ROCHESTER. Downloaded on September 15,2021 at 21:46:04 UTC from IEEE Xplore. Restrictions apply.