GTPC: Towards a Hybrid OLTP-OLAP Graph Benchmark

Muhammad Attahir Jibril[1] Alexander Baumstark[2] Kai-Uwe Sattler[3]

Abstract: Graph databases are gaining increasing relevance not only for pure analytics but also for full transactional support. Business requirements are evolving to demand analytical insights on fresh transactional data, thereby triggering the emergence of graph systems for hybrid transactional-analytical graph processing (HTAP). In this paper, we present our ongoing work on GTPC, a hybrid graph benchmark targeting such systems, based on the TPC-C and TPC-H benchmarks.

Keywords: Benchmarking; Graph HTAP; Graph Databases

1 Introduction

With the ever-growing amount of data from various real-world applications that lend themselves to being modelled as graphs, graph databases are receiving more and more attention from both the industry and academia for efficiently managing and processing such graph data. Various enterprises driven by their respective markets such as social media, logistics, e-commerce etc., are capitalizing on graph analytics for decision-support purposes. On top of that, in reality, the graph data is hardly without update operations. In fact, enterprises aim at continuously acquiring fresh business insights in order to make crucial business decisions. Some graph databases e.g. Neo4j [Neo4j], TigerGraph [TigerGraph], Ultipa [Ultipa] etc. support analytical workloads in addition to transactional workloads [Be19]. Additionally, more work is being put into graph storage systems such as LiveGraph [Zh20] that support concurrent execution of transactional and analytical workloads – towards the development of hybrid transactional-analytical (HTAP) graph databases.

Despite the relevance of emerging HTAP graph database systems, we identify the lack of a hybrid benchmark with mixed workloads that aims at these systems. Although there exist hybrid OLTP-OLAP benchmarks for other data models e.g, the relational model, however, they cannot be used directly for graphs. One such benchmark is the CH-benCHmark [Co11], which is based on the TPC-C [TPC-C10] and TPC-H [TPC-H21] benchmarks. In this paper, we propose GTPC, a hybrid graph benchmark modelled on a *product* graph (as obtained in domains like e-commerce) by adapting the underlying concepts of the CH-benCHmark. Firstly, GTPC employs schema optimizations to convert the relational CH-benCHmark schema into a property graph schema. Secondly, it provides a data generator (presently

[1] TU Ilmenau, DBIS, Helmholtzplatz 5, 98693 Ilmenau, muhammad-attahir.jibril@tu-ilmenau.de
[2] TU Ilmenau, DBIS, Helmholtzplatz 5, 98693 Ilmenau, alexander.baumstark@tu-ilmenau.de
[3] TU Ilmenau, DBIS, Helmholtzplatz 5, 98693 Ilmenau, kus@tu-ilmenau.de

doi:10.18420/BTW2023-05

a modified version of the CH-benCHmark data generator) for generating product graphs. Thirdly, it transforms the CH-benCHmark queries into transactional and analytical graph queries. Lastly, it specifies a mixed workload of the formulated graph queries. Although relational databases would traditionally be the solution for product graphs (i.e. products, orders and transactions data modelled as graphs), however, interestingly, product graphs are the most popular among graphs that model non-human entities [Sa20]. There is a need for benchmarks targeting such graph data and workloads, as none of the existing property graph benchmarks addresses the issue [Sa20]. This would provide a metric(s) with which to compare the systems used for such product graphs. Moreover, it would allow obtaining empirical insights into why industry practitioners use graph systems for processing data that would otherwise be considered suitable for relational systems.

To summarize, the increasing relevance of executing mixed workloads using HTAP graph databases as well as the huge adoption of graphs in modelling product data by industry practitioners motivate the need for a hybrid OLTP-OLAP graph benchmark. Such a benchmark is important as it serves as a useful tool for testing and comparing systems in terms of HTAP metrics such as freshness and performance isolation of concurrently running transactional and analytical workloads. Therefore, we present our ongoing work on benchmarking graph HTAP systems. Our contributions in this paper are as follows:

- We propose GTPC, a hybrid graph benchmark with mixed OLTP-OLAP workloads on a synthetically generated product graph based on the TPC-C and TPC-H benchmarks.
- Using our Poseidon graph database [Ji21] as an example, we implement and run the GTPC benchmark[4] to show its effectiveness in testing graph systems' handling of mixed workloads and revealing the performance interplay between analytical and transactional graph query workloads executed concurrently in a multi-user setting.

2 Related work

There exist various benchmarking solutions for graph databases stressing different workloads such as subgraph pattern matching, recursive path queries etc. These solutions include HPC Scalable Graph Analysis Benchmark [HPC09], LSQB [Mh21], gMark [Ba17], Wat-Div [Al14] etc. All of these target specific use cases but without considering any OLTP. The rapid growth of social media led to the development of benchmarks modelling social networks, e.g. LinkBench [Ar13], Linked Data Benchmark Council (LDBC) Social Network Benchmark (SNB) [Er15; LDBC FinBench] etc. However, unlike GTPC, they do not consider HTAP. Moreover, they are built around social graphs while GTPC targets product graphs (see Sect. 1). The LDBC Financial Benchmark (FinBench) targets workloads for financial scenarios. It is still ongoing, with the analytical queries not yet specified. Nevertheless, it also does not consider HTAP.

[4] https://dbgit.prakinf.tu-ilmenau.de/code/gtpc-neo4j

None of the aforementioned graph benchmarks tackles our use case of a product-graph-based benchmark for HTAP graph databases. GTPC stresses graph systems with transactional updates and analytical queries concurrently on the same graph. Contrary to graph databases, there are HTAP benchmarks for relational databases such as CH-benCHmark [Co11] and HTAPBench [Co17]. Both aim at merging the transactional-workload-based TPC-C benchmark and the analytical-workload-based TPC-H benchmark into a hybrid benchmark. The mixed workloads are run concurrently on the same tables in the same database.

3 Benchmark Design

In the following, we describe the design of the GTPC benchmark, which adapts the CH-benCHmark to a property graph model and the corresponding mixed query workload.

3.1 Data Model

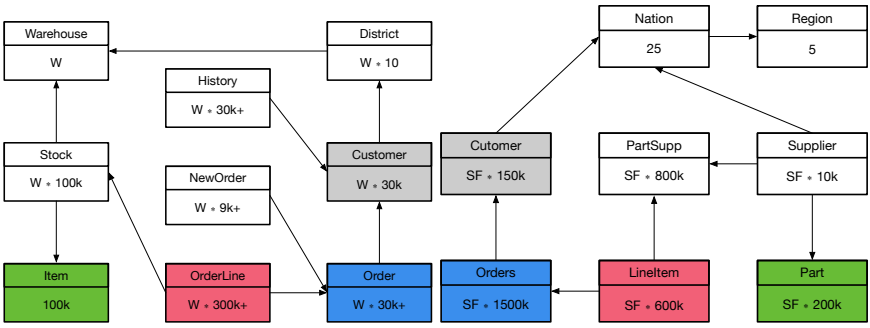
The product graph of GTPC is based on the Labeled Property Graph Model (or property graph) [An17]. The property graph model is widely adopted as it offers a rich representation of graphs where nodes (vertices) and relationships (edges) have types or *labels* and are associated with a set of properties stored as key-value pairs. In mathematical notation, a property graph G is a tuple

$$(N, R, sd, L, l_N, l_R, D, P_N, P_R)$$

where N is a finite set of nodes, $R \subseteq N \times N$ is a set of relationships, $sd : R \mapsto N \times N$ is a relationship function that maps each relationship to its source and destination nodes, L is a set of *labels* that define different types of nodes and relationships, $l_N : N \mapsto L$ and $l_R : R \mapsto L$ are labelling functions that assign types to nodes and relationships, $D = \cup_i D_i$ is the union of atomic domains D_i (since nodes and relationships may have an arbitrary number of properties), and P_N and P_R are sets of node and relationship properties respectively. A node property $p_i \in P_N$ is a partial function $p_i : N \mapsto D_i \cup \{NULL\}$, which assigns a property value from a domain $D_i \in D$ to a node $n \in N$ if n has the property p_i, otherwise $p_i(n)$ returns *NULL*. Similarly, a relationship property $p_j \in P_R$ is a partial function $p_j : R \mapsto D_j \cup \{NULL\}$, which assigns a property value from a domain $D_j \in D$ to a relationship $r \in R$ if r has the property p_j, otherwise $p_j(r)$ returns *NULL*.

3.2 Schema

The product graph of GTPC results from a merging of the TPC-C and TPC-H benchmarks, and their adaptation for graphs. Merging of TPC-C and TPC-H schemas had been done prior with the CH-benCHmark [Co11], a mixed workload benchmark for relational databases. We adopt some of its design considerations in GTPC.



(a) TPC-C (b) TPC-H

Fig. 1: (a) TPC-C Schema (b) TPC-H Schema.

TPC-C. The TPC-C simulates a general complex OLTP environment by way of five read-only and update-intensive transactions for the management, sale and distribution of a product or service. Specifically, it models a wholesale supplier having a certain number of warehouses and respective sales districts. The TPC-C schema covers nine relations as depicted in Fig. 1a, each associated with an entity of the database: WAREHOUSE, DISTRICT, CUSTOMER, ORDER, ORDER-LINE, ITEM, STOCK, NEW-ORDER and HISTORY. The transactions are New-Order, Payment, Order-Status, Delivery, and Stock-Level. In Fig. 1a, the arrows point in the direction of many-to-one relationships between the tables. The numbers denote the table cardinalities (the number of rows) and are expressed as factors of W, the number of warehouses, to show the scaling of the database. The "+" sign means that the number is subject to small variations as the number of rows changes [TPC-C10].

TPC-H. The TPC-H on the other hand simulates a business analytics application. It consists of 22 analytical queries for operations such as pricing, shipping management and market study. The business queries are executed on 8 relations: REGION, NATION, SUPPLIER, CUSTOMER, ORDER, LINEITEM, PART and PARTSUPP, as shown in Fig. 1b. Similarly to Fig. 1a, the arrows in Fig. 1b point in the direction of many-to-one table relationships and the numbers represent the cardinalities of the tables, expressed as factors of SF, the Scale Factor, which determines the database size [TPC-H21]. As we are adopting aspects of the CH-benCHmark, we use TPC-H instead of TPC-DS [TPC-DS21]. Moreover, unlike TPC-H, TPC-DS has a snowflake schema with multiple dimension and fact tables. It also incorporates an Extract-Transform-Load (ETL) process, which is contrary to HTAP systems. And with regards to having a unified schema for the mixed workloads, the similarities between the TPC-C schema and the TPC-H schema make TPC-H more suitable.

GTPC. GTPC combines the entities of TPC-C and TPC-H, where the entities are transformed into nodes. Both TPC-C and TPC-H share CUSTOMER and ORDER as common

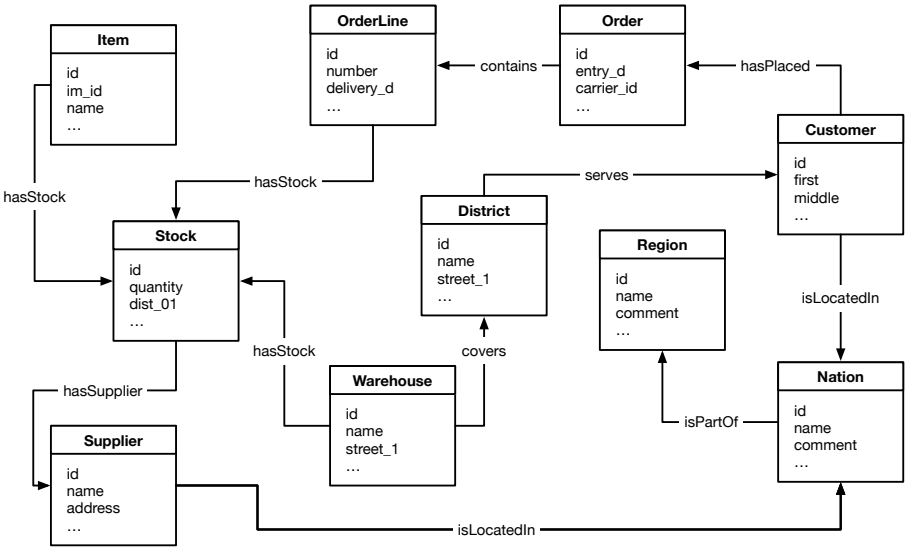


Fig. 2: GTPC Schema.

entities. Also, ORDERLINE and ITEM entities of TPC-C map to LINEITEM and PART entities of TPC-H respectively. We thus maintain the entities of TPC-C as base entities and incorporate the remaining SUPPLIER, REGION and NATION entities of TPC-H which are used solely for the analytical queries. By incorporating NATION, there is a need to introduce a relationship function *sd* that associates customers to their respective nations. Similarly to the CH-benCHmark, we take it from the first character of the *state* property of a CUSTOMER, which has 62 distinct values. Thus, we top up the 25 nations of the TPC-H to 62 nations.

We introduce identifiers for all node entities. This is required since, unlike in the relational model where entities contain foreign key attributes to denote relationships, nodes in a property graph do not store reference properties to other nodes. Each and every node of any given *label* is associated with an identifier that is unique to it within nodes of the same label. However, the identifiers are not unique across different labels. Nevertheless, each node is identifiable irrespective of the nodes to which it is connected, obviating the typical identifiers that are composites of foreign keys in the relational model.

Five out of the eight attributes of the HISTORY entities in TPC-C are foreign key attributes. As the benchmark does not require the history entities to be uniquely identifiable, we simply merge the remaining three attributes into the CUSTOMER entity in GTPC. As a result, HISTORY is not a separate entity in GTPC. This is a property graph schema optimization step that we take to avoid extra relationship traversal when retrieving history information. Additionally, we save space because the history data are simply stored as property key-values instead of as entire node objects [Al21]. Similarly, we do not store NEW-ORDER as separate node entities. And coupled with the fact that there are no attributes associated

with NEW-ORDER entities, we simply extend the properties of ORDER nodes with an extra property that indicates whether or not an ORDER node is new. The graph model elevates relationships to *first-class entities*. We thus model all relationships between all pairs of node entities as separate relationship entities. Fig. 2 depicts the resulting GTPC schema.

Note that *Part_Supp* in the TPC-H schema is an instance of a PART entity supplied by a certain SUPPLIER. In the TPC-C schema however, STOCK is an instance of an ITEM entity available in a certain WAREHOUSE. Since ITEM maps to PART, it follows that the information in *Part_Supp* in the TPC-H schema is analogous to that in STOCK in the GTPC schema. Hence, there is a relationship between SUPPLIER and STOCK in the GTPC schema (SUPPLIER does not exist is the TPC-C schema) much like there is a foreign key relationship between SUPPLIER and *Part_Supp* entity.

Future work includes addition of more graph features in the schema, e.g. self-edges. Self-edges could be introduced via SUPPLIER to SUPPLIER edges where bigger suppliers supply to smaller ones, via merging the REGION and NATION entity types into a single entity type, thereby transforming the existing regular edges with label `isPartOf` into self-edges etc.

3.3 Data Generation

Product graphs in GTPC are generated as property graphs based on the GTPC schema.

Generator: The GTPC graph generator is adapted from the CH-benCHmark data generator [CH]. Different graph generators employ various distributions such as power-law, Zipfian, uniform etc [Bo20]. As a first step, we simply follow the original TPC specification and adapt it in terms of assigning node degrees and property values, i.e. based on a uniform distribution. The graph data sets are output as CSV files.

Scaling: We use the number of warehouses as the basic scaling unit, similar to TPC-C. It determines the total number of nodes and other graph characteristics such as the degrees of all nodes, with the exception of ITEM, SUPPLIER, REGION and NATION.

3.4 Query Workloads

GTPC stresses the execution of concurrent OLTP and OLAP graph workloads. GTPC's OLTP graph workload is an adaptation of the five TPC-C transactions while its OLAP graph workload is an adaptation of the 22 TPC-H queries. We implement the OLTP and OLAP queries in C++ using the set of operators provided by Poseidon[5]. For other graph systems, the queries simply need to be implemented in the corresponding query language. As an example, we show the implementation of the GTPC OLAP #4 for Neo4j in List. 1.

[5] `https://dbgit.prakinf.tu-ilmenau.de/code/poseidon_core`

OLTP: GTPC OLTP queries 1–5 are transformations of the New-Order, Payment, Order-Status, Delivery, and Stock-Level transactions respectively into graph queries.

OLTP #1: This read-write transaction inserts an ORDER node along with a number of ORDERLINE nodes associated with it. The association is captured by adding a `contains` relationship for each ORDERLINE node, with the ORDER and ORDERLINE nodes as source and destination nodes respectively. Note that no NEW-ORDER nodes are created. Rather, the newly created ORDER nodes have one of their properties set to indicate they are new orders. Also note that additional relationships are created to connect the inserted ORDER node with its respective CUSTOMER node, and to connect the inserted ORDERLINE nodes to their respective STOCK nodes. The transaction also entails other read and write operations like retrieving WAREHOUSE, DISTRICT, CUSTOMER and STOCK nodes, as well as projecting some of their properties; and updating the properties of DISTRICT and STOCK nodes.

OLTP #2: This transaction updates a customer's balance. The transaction additionally updates the WAREHOUSE and DISTRICT nodes so that the sales tally with the new payment.

OLTP #3: This read-only transaction checks the order status of a customer. It covers retrieval of the CUSTOMER node, a relationship traversal(s) to retrieve the ORDER node(s), and a further relationship traversal(s) to retrieve the ORDERLINE node(s) connected to it.

OLTP #4: This read-write transaction delivers a batch of 10 new orders. It consists in retrieving the ORDER node and updating its properties to signify that it has been delivered and thus no longer a new order. Besides that, in order to reflect the delivery, its relationships are traversed to retrieve and update the delivery date properties of its associated ORDERLINE nodes; as well as the balance and delivery count of its associated CUSTOMER node.

OLTP #5: This read-only transaction computes the number of stock items sold recently and having a stock level below a certain threshold value. The transaction entails relationship traversals of up to four levels and an aggregate operation (count).

OLAP: We adapt the 22 queries of TPC-H for graphs, resulting in GTPC's OLAP queries 1–22. The TPC-H presents systems with a rich set of *chokepoints* or challenges with respect to optimized and efficient query processing. An analysis of the TPC-H chokepoints is presented in [BNE13]. The authors integrated chokepoints in their design of the LDBC benchmarks [LDBC SNB]. GTPC thus preserves those chokepoints. Additionally, since the OLAP queries are largely traversal operations ranging from one to eight hops, GTPC thus tests a system's ability to efficiently traverse the graph topology by choosing the optimal traversal order. This is central to the performance of graph processing.

Execution Mode: GTPC's mixed workload is run as concurrent streams of OLTP and OLAP queries. We dispatch an OLTP stream as a randomly permuted yet complete set of OLTP queries while an OLAP stream consists of the full set of OLAP queries ordered sequentially. Each OLAP stream starts with a different query. Each stream (OLTP or OLAP) is assigned a thread from a thread pool and, within the thread, the queries of the stream are

executed sequentially. Hence, the mixed execution mode consists in a mix of OLTP and OLAP streams that are dispatched in parallel from the thread pool and executed concurrently. As part of future work, an aspect of the execution to consider for fair comparison, especially as to systems with high OLTP performance, is bounding the graph size. This could be done by converting inserts into updates after a certain size limit or by using the number of OLTP streams that results in the maximum OLTP throughput.

```
MATCH(o:Order)-[:contains]->(ol:OrderLine)
WHERE o.entry_d >= datetime('2007-01-02T00:00:00.000000')
  AND o.entry_d < datetime('2012-01-02T00:00:00.000000')
  AND ol.delivery_d >= o.entry_d
WITH o.ol_cnt AS o_ol_cnt, COUNT(*) AS order_count
RETURN o_ol_cnt, order_count
```

List. 1: GTPC OLAP #4 in Cypher.

3.5 Benchmark Parameters

Our benchmark parameters are the database size in terms of the total number of nodes, which is a function of the number of warehouses; the number of concurrent OLTP and OLAP streams, which determines the level of contention between transactions; and the transaction isolation level, which determines transactional guarantees.

3.6 Performance Metrics

Most of the benchmarks discussed make use of execution time (latency) and/or throughput as performance metrics. Other metrics considered in benchmarking include CPU usage, memory footprint etc. In GTPC, we currently evaluate OLTP-OLAP performance interplay based on execution time and throughput.

4 Evaluation

We use our graph database, Poseidon [Ji21], as an example graph system for our evaluation in this paper. It should be noted here that although Poseidon is based on persistent memory, however, persistent memory is not relevant to GTPC. Poseidon is only a graph system we use here to make an example implementation of the GTPC benchmark. For concurrency control in this evaluation, we use the multi-version two-phase locking protocol, where the number of versions is limited to two (2V2PL).

We conduct our evaluations on a dual-socket Intel Xeon Gold 5215 with 10 cores per socket running at a maximum of 3.40 GHz. The machine is equipped with 384 GB DRAM, 1.5

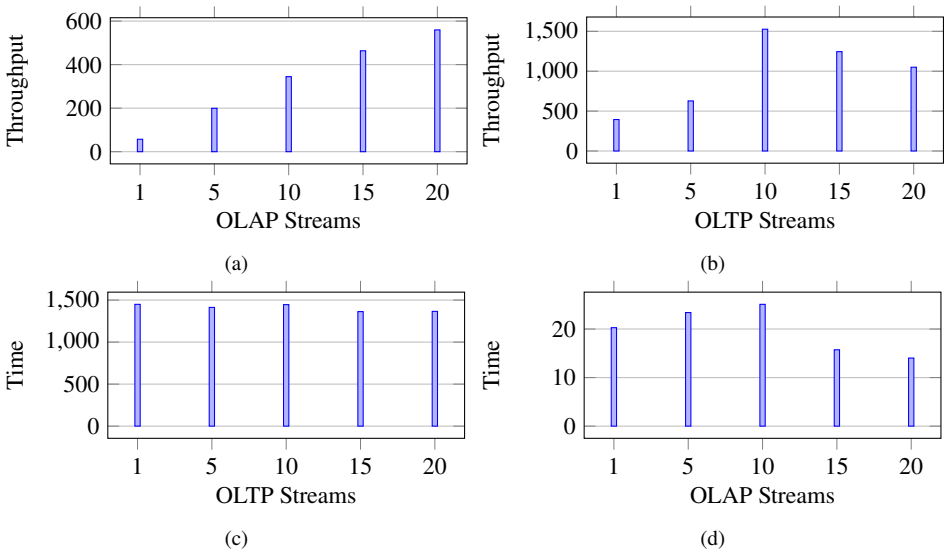


Fig. 3: (a) OLAP-only throughput (in Qph) with increasing number of OLAP streams (b) OLTP-only throughput (in Qph) with increasing number of OLTP streams (c) Execution time of OLAP stream (in sec) with increasing number of OLTP streams (d) Execution time of OLTP stream (in sec) with increasing number of OLAP streams.

TB Intel Optane DC Persistent Memory Module (DCPMM) operating in AppDirect mode, 4x 1.0 TB Intel SSD DC P4501 Series connected via PCIe 3.1; and runs on CentOS 7.9 with Linux Kernel 5.10.6. We use the Intel Persistent Memory Development Kit (PMDK) version 1.9.1 and libpmemobj-cpp version 1.11 for directly accessing the PMem device. Meanwhile, all executions were fixed to a single socket to factor out NUMA effects.

We load the GTPC dataset of two warehouses into Poseidon to execute the GTPC workloads. The input graph has 1.031.312 nodes and 1.992.528 relationships. We first execute OLAP streams exclusively. There is thus no contention for graph objects in this setting. Although the latency per OLAP stream increases with an increasing number of concurrent streams, Fig. 3a shows that the overall query throughput (expressed in queries per hour) increases. With OLTP-only stream execution, however, contention between transactions on graph objects results in some of them aborting. Also, different transactions abort at different stages of execution – with computation being wasted for each transaction that aborts. All these depend on the graph characteristics and workload pattern. More skew would result in higher contention on common graph objects, as transactions are more likely to access the same nodes – especially those with higher degrees. We currently adopt the TPC specification in our relationship mapping and substitution parameters. Nevertheless, we see in Fig. 3b that transaction throughput increases initially with an increase in concurrent OLTP streams until

it reaches a threshold at 10 streams, after which dispatching additional concurrent streams decreases throughput.

Thereafter, we run the mixed OLTP-OLAP workload to demonstrate the effectiveness of GTPC in testing performance isolation in a system – i.e. the system's ability to handle the interference between concurrently running OLTP and OLAP query streams without the performance of either of the two being compromised as a result of the other. We start by fixing the OLAP stream at a single stream while varying the concurrent OLTP streams. Fig. 3c shows the execution times (in seconds) of the individual queries in the OLAP stream when run concurrently with a varying number of OLTP streams. For each query, its execution times in the presence of the varying number of OLTP streams lie within a relatively narrow range. This shows that the OLTP streams do not interfere much with the OLAP query execution times, as the OLTP transactions do not block the OLAP queries due to multi-versioning. Compared with the earlier OLAP-only runs, we see that the execution time of an OLAP stream is influenced more by concurrent OLAP streams than by the same number of OLTP streams. Finally, we maintain a single OLTP stream and run it concurrently with a varying number of OLAP streams. We expect the OLTP stream execution time to increase with more OLAP streams partly because the improved concurrency of OLAP queries in the 2V2PL is a trade-off with delaying transaction commit when the OLTP transactions wait for OLAP queries before acquiring a *certify lock*. The initial part of Fig. 3d shows that. We note here that we omit the execution time of OLTP #4, which is the most write-heavy transaction after OLTP #1, as it fails starting from 10 OLAP streams.

5 Conclusion

In this paper, we have presented our ongoing work on GTPC, a hybrid OLTP-OLAP graph benchmark based on the TPC-C and TPC-H benchmarks. We implemented and ran GTPC on Poseidon, our graph database, as an example to showcase the effectiveness of GTPC in testing HTAP graph systems with respect to performance isolation between concurrently running OLTP and OLAP graph query streams. Future work towards developing GTPC into a fully-fledged hybrid graph benchmark include extending the mixed workload to further include graph algorithms, incorporating more real-world data characteristics in the data generation, introducing more benchmark parameters and performance metrics to facilitate better comparison between different HTAP graph systems.

Acknowledgements. This work was partially funded by the German Research Foundation (DFG) in the context of the project "Hybrid Transactional/Analytical Graph Processing in Modern Memory Hierarchies (#TAG)" (SA 782/28-2) as part of the priority program "Scalable Data Management for Future Hardware" (SPP 2037) and by the Carl-Zeiss-Stiftung under the project "Memristive Materials for Neuromorphic Electronics (MemWerk)".

References

[Al14] Aluç, G.; Hartig, O.; Özsu, M. T.; Daudjee, K.: Diversified Stress Testing of RDF Data Management Systems. In (Mika, P.; Tudorache, T.; Bernstein, A.; Welty, C.; Knoblock, C. A.; Vrandecic, D.; Groth, P.; Noy, N. F.; Janowicz, K.; Goble, C. A., eds.): The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I. Vol. 8796. Lecture Notes in Computer Science, Springer, pp. 197–212, 2014, URL: https://doi.org/10.1007/978-3-319-11964-9%5C_13.

[Al21] Alotaibi, R.; Lei, C.; Quamar, A.; Efthymiou, V.; Özcan, F.: Property Graph Schema Optimization for Domain-Specific Knowledge Graphs. In: 37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021. IEEE, pp. 924–935, 2021, URL: https://doi.org/10.1109/ICDE51399.2021.00085.

[An17] Angles, R.; Arenas, M.; Barceló, P.; Hogan, A.; Reutter, J. L.; Vrgoc, D.: Foundations of Modern Query Languages for Graph Databases. ACM Comput. Surv. 50/5, 68:1–68:40, 2017, URL: https://doi.org/10.1145/3104031.

[Ar13] Armstrong, T. G.; Ponnekanti, V.; Borthakur, D.; Callaghan, M.: LinkBench: a database benchmark based on the Facebook social graph. In (Ross, K. A.; Srivastava, D.; Papadias, D., eds.): Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013. ACM, pp. 1185–1196, 2013, URL: https://doi.org/10.1145/2463676.2465296.

[Ba17] Bagan, G.; Bonifati, A.; Ciucanu, R.; Fletcher, G. H. L.; Lemay, A.; Advokaat, N.: gMark: Schema-Driven Generation of Graphs and Queries. IEEE Trans. Knowl. Data Eng. 29/4, pp. 856–869, 2017, URL: https://doi.org/10.1109/TKDE.2016.2633993.

[Be19] Besta, M.; Peter, E.; Gerstenberger, R.; Fischer, M.; Podstawski, M.; Barthels, C.; Alonso, G.; Hoefler, T.: Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. CoRR abs/1910.09017/, 2019, arXiv: 1910.09017, URL: http://arxiv.org/abs/1910.09017.

[BNE13] Boncz, P. A.; Neumann, T.; Erling, O.: TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. In (Nambiar, R.; Poess, M., eds.): Performance Characterization and Benchmarking - 5th TPC Technology Conference, TPCTC 2013, Trento, Italy, August 26, 2013, Revised Selected Papers. Vol. 8391. Lecture Notes in Computer Science, Springer, pp. 61–76, 2013, URL: https://doi.org/10.1007/978-3-319-04936-6%5C_5.

[Bo20] Bonifati, A.; Holubová, I.; Prat-Pérez, A.; Sakr, S.: Graph Generators: State of the Art and Open Challenges. ACM Comput. Surv. 53/2, 36:1–36:30, 2020, URL: https://doi.org/10.1145/3379445.

[CH] CH-benCHmark, URL: https://db.in.tum.de/research/projects/CHbenCHmark/index.shtml.

[Co11] Cole, R. L.; Funke, F.; Giakoumakis, L.; Guy, W.; Kemper, A.; Krompass, S.; Kuno, H. A.; Nambiar, R. O.; Neumann, T.; Poess, M.; Sattler, K.; Seibold, M.; Simon, E.; Waas, F.: The mixed workload CH-benCHmark. In (Graefe, G.; Salem, K., eds.): Proceedings of the Fourth International Workshop on Testing Database Systems, DBTest 2011, Athens, Greece, June 13, 2011. ACM, p. 8, 2011, URL: https://doi.org/10.1145/1988842.1988850.

[Co17] Coelho, F.; Paulo, J.; Vilaça, R.; Pereira, J.; Oliveira, R.: HTAP-Bench: Hybrid Transactional and Analytical Processing Benchmark. In (Binder, W.; Cortellessa, V.; Koziolek, A.; Smirni, E.; Poess, M., eds.): Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017. ACM, pp. 293–304, 2017, URL: https://doi.org/10.1145/3030207.3030228.

[Er15] Erling, O.; Averbuch, A.; Larriba-Pey, J. L.; Chafi, H.; Gubichev, A.; Prat-Pérez, A.; Pham, M.; Boncz, P. A.: The LDBC Social Network Benchmark: Interactive Workload. In (Sellis, T. K.; Davidson, S. B.; Ives, Z. G., eds.): Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015. ACM, pp. 619–630, 2015, URL: https://doi.org/10.1145/2723372.2742786.

[HPC09] HPC Scalable Graph Analysis Benchmark, 2009, URL: http://www.graphanalysis.org/benchmark/GraphAnalysisBenchmark-v1.0.pdf.

[Ji21] Jibril, M. A.; Baumstark, A.; Götze, P.; Sattler, K.: JIT happens: Transactional Graph Processing in Persistent Memory meets Just-In-Time Compilation. In (Velegrakis, Y.; Zeinalipour-Yazti, D.; Chrysanthis, P. K.; Guerra, F., eds.): Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021. OpenProceedings.org, pp. 37–48, 2021, URL: https://doi.org/10.5441/002/edbt.2021.05.

[LDBC FinBench] The LDBC Financial Benchmark, URL: https://ldbcouncil.org/ldbc_finbench_docs/ldbc-finbench-specification.pdf.

[LDBC SNB] The LDBC Social Network Benchmark, URL: http://ldbc.github.io/ldbc_snb_docs/ldbc-snb-specification.pdf.

[Mh21] Mhedhbi, A.; Lissandrini, M.; Kuiper, L.; Waudby, J.; Szárnyas, G.: LSQB: a large-scale subgraph query benchmark. In (Kalavri, V.; Yakovets, N., eds.): GRADES-NDA '21: Proceedings of the 4th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA), Virtual Event, China, 20 June 2021. ACM, 8:1–8:11, 2021, URL: https://doi.org/10.1145/3461837.3464516.

[Neo4j] Neo4j, URL: https://neo4j.com/.

[Sa20] Sahu, S.; Mhedhbi, A.; Salihoglu, S.; Lin, J.; Özsu, M. T.: The ubiquity of large graphs and surprising challenges of graph processing: extended survey. VLDB J. 29/2-3, pp. 595–618, 2020, URL: https://doi.org/10.1007/s00778-019-00548-x.

[TigerGraph] TigerGraph, URL: https://www.tigergraph.com.

[TPC-C10] TPC-C Specification, 2010, URL: http://tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf.

[TPC-DS21] TPC-DS specification, 2021, URL: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v3.2.0.pdf.

[TPC-H21] TPC-H Specification, 2021, URL: http://tpc.org/tpc_documents_current_versions/pdf/tpc-h_v3.0.0.pdf.

[Ultipa] Ultipa, URL: https://www.ultipa.com/.

[Zh20] Zhu, X.; Serafini, M.; Ma, X.; Aboulnaga, A.; Chen, W.; Feng, G.: LiveGraph: A Transactional Graph Storage System with Purely Sequential Adjacency List Scans. Proc. VLDB Endow. 13/7, pp. 1020–1034, 2020, URL: http://www.vldb.org/pvldb/vol13/p1020-zhu.pdf.