Benchmarking Hybrid OLTP&OLAP Database Systems

Florian Funke Alfons Kemper Thomas Neumann
{first.last}@in.tum.de

Technische Universität München
Fakultät für Informatik
Boltzmannstr. 3
85748 Garching, Germany

Abstract: Recently, the case has been made for operational or real-time Business Intelligence (BI). As the traditional separation into OLTP database and OLAP data warehouse obviously incurs severe latency disadvantages for operational BI, hybrid OLTP&OLAP database systems are being developed. The advent of the first generation of such hybrid OLTP&OLAP database systems requires means to characterize their performance.

While there are standardized and widely used benchmarks addressing either OLTP or OLAP workloads, the lack of a hybrid benchmark led us to the definition of a new mixed workload benchmark, called TPC-CH. This benchmark bridges the gap between the existing single-workload suits: TPC-C for OLTP and TPC-H for OLAP. The newly proposed TPC-CH benchmark executes a mixed workload: A transactional workload based on the order entry processing of TPC-C and a corresponding TPC-H-equivalent OLAP query suite on this sales data base. As it is derived from these two most widely used TPC benchmarks our new TPC-CH benchmark produces results that are highly comparable to both, hybrid systems and classic single-workload systems. Thus, we are able to compare the performance of our own (and other) hybrid database system running both OLTP and OLAP workloads in parallel with the OLTP performance of dedicated transactional systems (e.g., VoltDB) and the OLAP performance of specialized OLAP databases (e.g., column stores such as MonetDB).

1 Introduction

The two areas of online transactions processing (OLTP) and online analytical processing (OLAP) constitute different challenges for database architectures. While transactions are typically short-running and perform very selective data access, analytical queries are generally longer-running and often scan significant portions of the data. Therefore customers with high rates of mission-critical transactions are currently forced to operate two separate systems: one operational database processing transactions and one *data warehouse* dedicated to analytical queries. The data warehouse is periodically updated with data that is extracted from the OLTP system and transformed into a schema optimized for analysis. Early attempts to run analyses directly on the operational systems resulted in unacceptable transaction processing performance [DHKK97].

While this data staging approach allows each system to be tuned for its respective workload, it suffers from several inherent drawbacks: Two software and hardware systems must be purchased and maintained. Additional systems may be required depending on the data staging implementation. All systems have to store redundant copies of the same data, but most importantly, analyses do not incorporate the latest data, but work on the stale snapshot in the data warehouse.

Recently, the case has been made for so called *real-time Business Intelligence*. SAP's cofounder Hasso Plattner [Pla09] criticizes the separation between OLTP and OLAP deploring a shift of priorities towards OLTP. He emphasizes the necessity of OLAP for strategic management and compares the expected impact of real-time analysis on management with the impact of Internet search engines on the world.

Real-time business intelligence postulates novel types of database architectures, often based on in-memory technology, such as the Hybrid Row-Column OLTP Database Architecture for Operational Reporting [SBKZ08, BHF09, KGT+10] or HyPer [KN11]. They address both workloads with a single system, thereby eliminating the aforementioned shortcomings of the data staging approach.

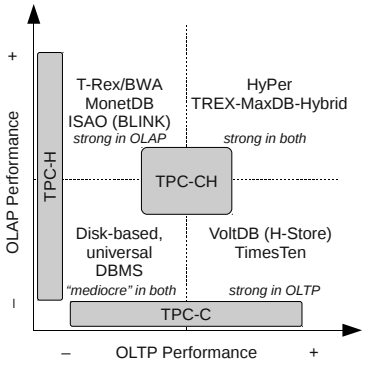


Figure 1: Classification of DBMS and Benchmarks

Different strategies seem feasible to reconcile frequent inserts and updates with longer running BI queries: Modifications triggered by transactions may be collected in a delta and periodically merged with the main dataset which serves as a basis for queries [KGT+10]. Alternatively, the DBMS can devise versioning to separate transaction processing on the latest version from queries operating on a snapshot of the versionized data.

This novel class of DBMS necessitates means to analyze their performance. Hybrid systems need to be compared against each other to evaluate the different implementation strategies. They must also be juxtaposed to traditional, universal DBMS and specialized single-workload systems to prove their competitiveness in terms of performance and resource consumption.

We present TPC-CH, a benchmark that seeks to produce highly comparable results for all types of systems (cf. Figure 1). The following section evaluates related benchmarks. Section 3 describes the design of the TPC-CH. Section 4 describes the systems under test.

Section 5 shows setups and results produced with different types of DBMS and Section 6 concludes the paper.

2 Related Work

The Transaction Processing Performance Council (TPC) specifies benchmarks that are widely used in industry and academia to measure performance characteristics of database systems. TPC-C and its successor TPC-E simulate OLTP workloads. The TPC-C schema consists of nine relations and five transactions that are centered around the management, sale and distribution of products or services. The database is initially populated with random data and then updated as new orders are processed by the system. TPC-E simulates the workload of a brokerage firm. It features a more complex schema and pseudo-real content that seeks to match actual customer data better. However, TPC-C is far more pervasive compared to TPC-E [Tra10c, Tra10d] and thus offers better comparability.

TPC-H is currently the only active decision support benchmark of the TPC. It simulates an analytical workload in a business scenario similar to TPC-C's. The benchmark specifies 22 queries on the 8 relations that answer business questions. TPC-DS, its dedicated successor, will feature a star-schema, around 100 decision support queries and a description of the ETL process that populates the database. However, it currently is in draft state.

Note that composing a benchmark for hybrid DBMS by simply using two TPC schemas, one for OLTP and one for OLAP, does not produce meaningful results. Such a benchmark would not give insight into how a system handles its most challenging task: The concurrent processing of transactions and queries on the *same* data.

The composite benchmark for online transaction processing (CBTR) [BKS08] was proposed to measure the impact of a workload that comprises both OLTP and operational reporting. CBTR is no combination of existing standardized benchmarks, but uses an enterprises' real data. The authors mention the idea of a data generator to produce results that allow comparisons between systems. Yet the focus of CBTR seems to be the comparison of different database systems for the specific use case of a certain enterprise.

3 Benchmark Design

Our premier goal in the design of TPC-CH was comparability. Therefore, we leverage a combination of TPC-C and TPC-H. Both benchmarks are widely used and accepted, relatively fast to implement and have enough similarity in their design to make a combination possible.

TPC-CH is comprised of the unmodified TPC-C schema and transactions and an adapted version of the TPC-H queries. Since the schemas of both benchmarks (cf. Figure 2) model businesses which "must manage, sell, or distribute a product or service" [Tra10a, Tra10b], they have some similarities between them. The relations ORDER(S) and CUSTOMER exist

in both schemas. Moreover, both ORDER-LINE (TPC-C) and LINEITEM (TPC-H) model entities that are sub-entities of ORDER(S) and thus resemble each other.

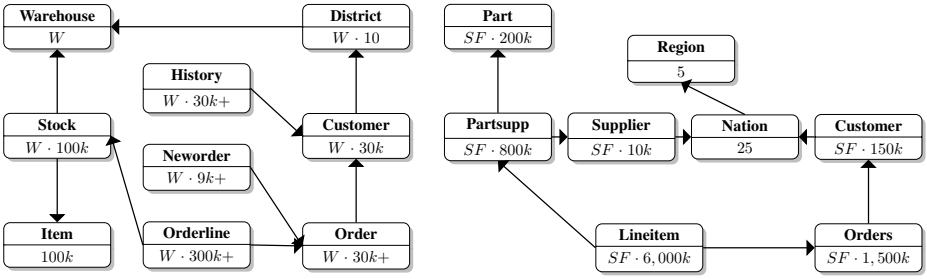


Figure 2: TPC-C and TPC-H schemas

TPC-CH keeps all TPC-C entities and relationships completely unmodified and integrates the likewise unchanged relations SUPPLIER, REGION and NATION from the TPC-H schema. These relations are frequently used in TPC-H queries and allow a non-intrusive integration into the TPC-C schema. The relation SUPPLIER is populated with a fixed number (10,000) of entries. Thereby, an entry in STOCK can be uniquely associated with its SUPPLIER through the relationship STOCK.S_I_ID $\times$ STOCK.S_W_ID $mod\ 10,000 =$ SUPPLIER.SU_SUPPKEY.

The original TPC-C relation CUSTOMER contains no foreign key that references the associated NATION. Since we keep the original schema untouched to preserve compatibility with existing TPC-C installations, the foreign key is computed from the first character of the field C_STATE. TPC-C specifies that this first character can have 62 different values (upper-case letters, lower-case letters and numbers), therefore we chose 62 nations to populate NATION. The primary key N_NATIONKEY is an identifier according to the TPC-H specification. Its values are chosen such that their associated ASCII value is a letter or number (i.e. N_NATIONKEY $\in [48, 57] \cup [65, 90] \cup [97, 122]$). Therefore no additional calculations are required to skip over the gaps in the ASCII code between numbers, upper-case letters and lower-case letters. Database systems that do not provide a conversion routine from a character to its ASCII code may deviate from the TPC-H schema and use a single character as a primary key for NATION. REGION contains the five regions of these nations. Relationships between the new relations are modeled with simple foreign key fields (NATION.N_REGIONKEY and SUPPLIER.SU_NATIONKEY).

3.1 Transactions and Queries

As illustrated in the overview in Figure 4, the workload consists of the five original TPC-C transactions and 22 queries adopted from TPC-H. Since the TPC-C schema is an unmodified subset of the TPC-CH schema, the original transactions can be executed without any modification:

New-Order This transaction enters an order with multiple order-lines into the database. For each order-line, 99% of the time the supplying warehouse is the home ware-

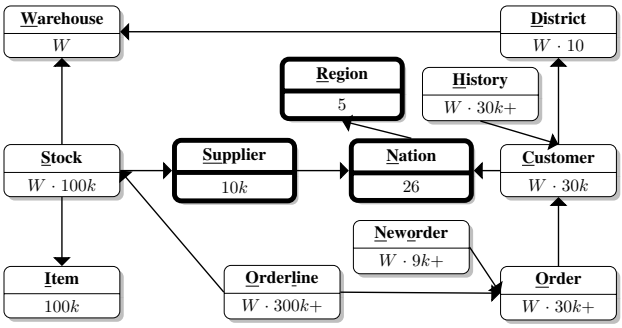


Figure 3: TPC-CH schema. Entities originating from TPC-H are highlighted.

house. The home warehouse is a fixed warehouse ID associated with a terminal. To simulate user data entry errors, 1% of the transactions fail and trigger a roll-back.

Payment A payment updates the balance information of a customer. 15% of the time, a customer is selected from a random remote warehouse, in the remaining 85%, the customer is associated with the home warehouse. The customer is selected by last name in 60% of the cases and else by his three-component key.

Order-Status This read-only transaction is reporting the status of a customer's last order. The customer is selected by last name 60% of the time. If not selected by last name, he is selected by his ID. The selected customer is always associated with the home warehouse.

Delivery This transaction delivers 10 orders in one batch. All orders are associated with the home warehouse.

Stock-Level This read-only transaction operates on the home warehouse only and returns the number of those stock items that were recently sold and have a stock level lower than a threshold value.

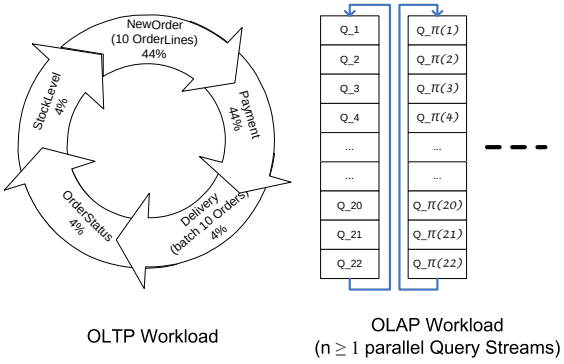


Figure 4: Benchmark overview: OLTP and OLAP on the same data

The distribution over the five transaction types conforms to the TPC-C specification (cf. Figure 4), resulting in frequent execution of the New-Order and Payment transactions.

TPC-CH deviates from the underlying TPC-C benchmark by not simulating the terminals and by generating client requests without any think-time, as proposed by [Vola]. This way, very high transaction rates can be achieved on relatively small database configurations. Since the transactions themselves remain the same as in TPC-C, TPC-CH results are directly comparable to existing TPC-C results with the same modifications, e.g. VoltDB [Vola]. Moreover, these changes can be easily applied to existing TPC-C implementations in order to produce TPC-CH-compatible results.

For the OLAP portion of the workload, we adopt the 22 queries from TPC-H to the TPC-CH schema. In reformulating the queries to match the slightly different schema, we made sure that their business semantics and syntactical structure were preserved. E.g., query 5 lists the revenue achieved through local suppliers (cf. Listing 1 and 2). Both queries join similar relations, have similar selection criteria, perform summation, grouping and sorting.

```
SELECT n_name, SUM(ol_amount) AS revenue
FROM customer, "order", orderline, stock, supplier, nation, region
WHERE c_id=o_c_id AND c_w_id=o_w_id AND c_d_id=o_d_id
  AND ol_o_id=o_id AND ol_w_id=o_w_id AND ol_d_id=o_d_id
  AND ol_w_id=s_w_id AND ol_i_id=s_i_id
  AND mod((s_w_id * s_i_id),10000)=su_suppkey
  AND ascii(SUBSTRING(c_state, 1, 1))=su_nationkey
  AND su_nationkey=n_nationkey AND n_regionkey=r_regionkey
  AND r_name='[REGION]' AND o_entry_d>='[DATE]'
GROUP BY n_name ORDER BY revenue DESC
```

Listing 1: TPC-CH Query 5

```
SELECT n_name, SUM(l_extendedprice * (1 - l_discount)) AS revenue
FROM customer, orders, lineitem, supplier, nation, region
WHERE c_custkey = o_custkey AND l_orderkey = o_orderkey
  AND l_suppkey = s_suppkey AND c_nationkey = s_nationkey
  AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey
  AND r_name = '[REGION]' AND o_orderdate >= DATE '[DATE]'
  AND o_orderdate < DATE '[DATE]' + INTERVAL '1' YEAR
GROUP BY n_name ORDER BY revenue DESC
```

Listing 2: TPC-H Query 5

TPC-CH does not require refresh functions, as specified in TPC-H, since the TPC-C transactions are continuously updating the database. The following section particularizes, when queries have to incorporate these updates.

3.2 Benchmark Parameters

TPC-CH has four scales: First, the database size is variable. As in TPC-C, the size of the database is specified through the number of warehouses. Most relations grow with the number of warehouses, with Item, Supplier, Nation and Region being the only ones of constant size.

The second scale is the composition of the workload. It can be comprised of analytical queries only, transactions only or any combination of the two. The workload mix is specified as the number of parallel OLTP and OLAP sessions (streams) that are connected to the database system. An OLTP session dispatches random TPC-C transactions sequentially with the distribution described in the official specification [Tra10a]. An analytical session performs continuous iterations over the query set which is comprised of all 22 queries. Each session starts with a different query to avoid caching effects between sessions as depicted in Figure 4.

The third input parameter is the isolation level. Lower isolation levels like read committed allow for faster processing, while higher isolation levels guarantee higher quality results for both transactions and queries.

Finally, the freshness of the data that is used as a basis for the analyses is a parameter of the benchmark. It only applies if the workload mix contains both, OLTP and OLAP components. Data freshness is specified as the time or the number of transactions after which newly issued query sets have to incorporate the most recent data. This allows for both, database architectures that have a single dataset for both workloads and those that devise a delta to run the benchmark.

3.3 Reporting requirements

In addition to a description of the hard- and software employed, the following characteristics of the system are reported: The OLTP engine's performance is quantified by the throughput of New-Order transactions and all transactions. On the OLAP side, the query response time is measured for each query in each iteration and session. The median value and the query throughput are reported. In addition to the freshness parameter, the maximum dataset age is reported. For in-memory systems, the total memory consumption of all processes over time is reported. This includes allocated, but not yet used memory chunks. Figure 5 shows a sample report.

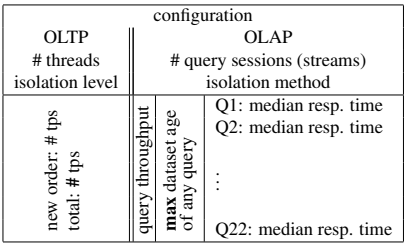


Figure 5: Reporting Requirements of the TPC-CH Benchmark

4 Systems under Test

In Figure 1, we grouped DBMSs in four segments. We use TPC-CH to analyze the performance of one representative of each category.

4.1 OLAP-focused Database Systems

MonetDB is the most influential database research project on column store storage schemes for in-memory OLAP databases. An overview of the system can be found in the summary paper [BMK09] presented on the occasion of receiving the 10 year test of time award of the VLDB conference. Therefore, we use MonetDB as a representative of the "strong in OLAP"-category. Other systems in this category are TREX (BWA) of SAP, IBM Smart Analytics Optimizer and Vertica Analytic Database.

4.2 OLTP-focused Database Systems

The H-Store prototype [KKN+08], created by researchers led by Michael Stonebraker was recently commercialized by a start-up company name VoltDB. VoltDB is a high-performance, in-memory OLTP system that pursues a lock-less approach [HAMS08] to transaction processing where transactions operate on private partitions and are executed in serial [Volb]. VoltDB represents the "strong in OLTP"-category. This category also includes the following systems: P*Time [CS04], IBM solidDB, TimesTen of Oracle and the new startup developments Electron DB, Clustrix, Akiban, dbShards, NimbusDB, ScaleDB and Lightwolf.

4.3 Universal Database Systems

This category contains the disk-based, universal database systems. We picked a popular, commercially available one ("System X") as a representative of the universal DBMS category.

4.4 Hybrid Database Systems

This category includes the new database development at SAP as outlined by Hasso Plattner [Pla09] and our HyPer [KN11] system. A special-purpose OLTP&OLAP system is Crescando [GUMG10] that has, however, limited query capabilites.

4.4.1 HyPer: Virtual Memory Snapshots

We have developed a novel hybrid OLTP&OLAP database system based on snapshotting transactional data via the virtual memory management of the operating system [KN11]. In this architecture the OLTP process "owns" the database and periodically (e.g., in the order of seconds or minutes) forks an OLAP process. This OLAP process constitutes a fresh transaction consistent snapshot of the database. Thereby, we exploit the operating

systems functionality to create virtual memory snapshots for new, duplicated processes. In Unix, for example, this is done by creating a child process of the OLTP process via the `fork()` system call. To guarantee transactional consistency, the `fork()` should only be executed in between two (serial) transactions, never in the middle of one transaction. In section 4.4.1 we will relax this constraint by utilizing the undo log to convert an action consistent snapshot (created in the middle of a transaction) into a transaction consistent one.

The forked child process obtains an exact copy of the parent processes address space, as exemplified in Figure 6 by the overlayed page frame panel. This virtual memory snapshot that is created by the `fork()`-operation will be used for executing a session of OLAP queries – as indicated on the right hand side of Figure 6.

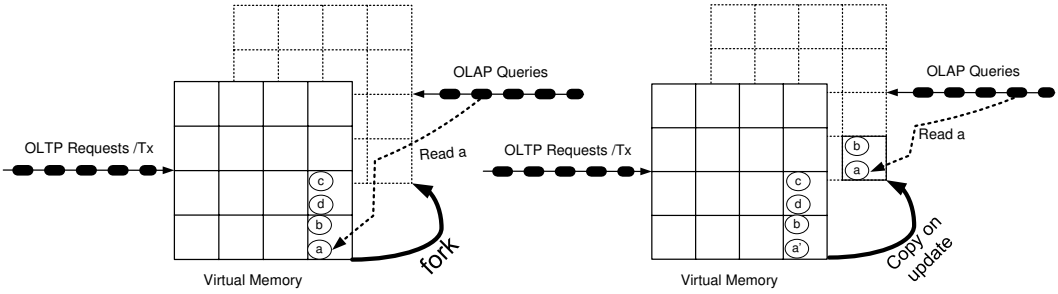


Figure 6: Forking a new Snapshot (left) and copy-on-update/write (right)

The snapshot stays in precisely the state that existed at the time the `fork()` took place. Fortunately, state-of-the-art operating systems do not physically copy the memory segments right away. Rather, they employ a lazy *copy-on-update* strategy – as sketched out in Figure 6. Initially, parent process (OLTP) and child process (OLAP) share the same physical memory segments by translating either virtual addresses (e.g., to object a) to the same physical main memory location. The sharing of the memory segments is highlighted in the graphics by the dotted frames. A dotted frame represents a virtual memory page that was not (yet) replicated. Only when an object, like data item a, is updated, the OS- and hardware-supported copy-on-update mechanism initiate the replication of the virtual memory page on which a resides. Thereafter, there is a new state denoted a' accessible by the OLTP-process that executes the transactions and the old state denoted a, that is accessible by the OLAP query session. Unlike the figure suggests, the additional page is really created for the OLTP process that initiated the page change and the OLAP snapshot refers to the old page – this detail is important for estimating the space consumption if several such snapshots are created (cf. Figure 7).

So far we have sketched a database architecture utilizing two processes, one for OLTP and another one for OLAP. As the OLAP queries are *read-only* they could easily be executed in parallel in multiple threads that share the same address space. Still, we can avoid any synchronization (locking and latching) overhead as the OLAP queries do not share any mutable data structures. Modern multi-core computers which typically have more than ten cores can certainly yield a substantial speed up via this inter-query parallelization.

Another possibility to make good use of the multi-core servers is to create multiple snapshots. The HyPer architecture allows for arbitrarily current snapshots. This can simply be

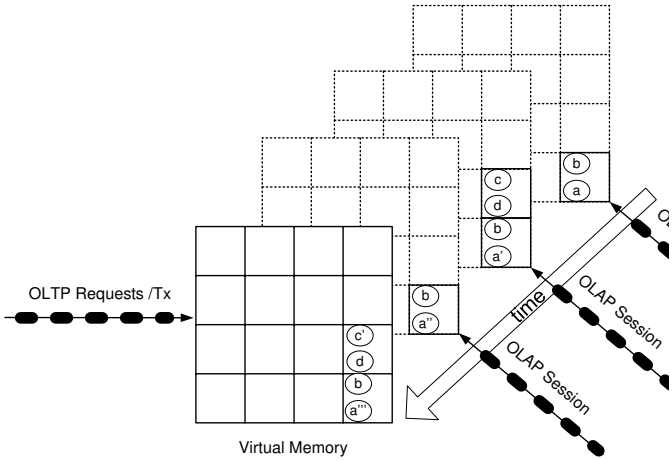


Figure 7: Multiple OLAP Sessions at Different Points in Time

achieved by periodically (or on demand) `fork()`-ing a new snapshot and thus starting a new OLAP query session process. This is exemplified in Figure 7. Here we sketch the one and only OLTP process'es current database state (the front panel) and three active query session processes' snapshots – the oldest being the one in the background. The successive state changes are highlighted by the four different states of data item a (the oldest state), a', a'', and a''' (the youngest transaction consistent state). Obviously, most data items do not change in between different snapshots as we expect to create snapshots for most up-to-date querying at intervals of a few seconds – rather than minutes or hours as is the case in current separated data warehouse solutions with ETL data staging. The number of active snapshots is, in principle, not limited, as each "lives" in its own process. By adjusting the priority we can make sure that the mission critical OLTP process is always allocated a core – even if the OLAP processes are numerous and/or utilize multi-threading and thus exceed the number of cores.

A snapshot will be deleted after the last query of a session is finished. This is done by simply terminating the process that was executing the query session. It is not necessary to delete snapshots in the same order as they were created. Some snapshots may persist for a longer duration, e.g., for detailed stocktaking purposes. However, the memory overhead of a snapshot is proportional to the number of transactions being executed from creation of this snapshot to the time of the next younger snapshot (if it exists or to the current time). The figure exemplifies this on the data item c which is physically replicated for the "middle age" snapshot and thus shared and accessible by the oldest snapshot. Somewhat against our intuition, it is still possible to terminate the middle-aged snapshot before the oldest snapshot as the page on which c resides will be automatically detected by the OS-/processor as being shared with the oldest snapshot via a reference counter associated with the physical page. Thus it survives the termination of the middle-aged snapshot – unlike the page on which a' resides which is freed upon termination of the middle-aged snapshot process. The youngest snapshot accesses the state c' that is contained in the current OLTP process'es address space.

5 Results

In this section, we present cursory results with TPC-CH. Our experiments are performed on a machine with two quad-core 2.93 GHz Intel® Xeon® processors and 64GB of memory running Red Hat Enterprise Linux 5.4. All databases are scaled to 12 warehouses and we performed 5 iterations over the query sets.

For MonetDB, we evaluate an instance of the benchmark that performs a pure-OLAP workload. We excluded OLTP because the absence of indexes in MonetDB prevents efficient transaction processing. We present results of setups with three parallel OLAP sessions in Figure 9. Since there are no updates to the database in this scenario, the freshness and the isolation level parameter are lapsed. Increasing the number of query streams to 5 hardly changes the throughput, but almost doubled the query execution times. Running a single query session improved the execution times between 10% and 45% but throughput deteriorates to 0.55 queries per second.

For VoltDB, the workload-mix includes transactions only. One "site" per warehouse/-partition (i.e. 12 sites) yields best results on our server. Differing from the TPC-CH specification, we allow VoltDB to execute only single-partition transactions, as suggested in [Vola] and skip those instances of New-Order and Payment that involve more than one warehouse. The isolation level in VoltDB is serializable.

For System X, we use 25 OLTP sessions and 3 OLAP sessions. The configured isolation level is read committed for both OLTP and OLAP and we use group committing with groups of five transactions. Since the system operates on a single dataset, every query operates on the latest data. Figure 9 shows the results of this setup. Increasing the OLAP sessions from 3 to 12 enhances the query throughput from 0.38 to 1.20 queries/s, but causes the query execution times to go up by 20 to 30% and a decline of the OLTP throughput by 14%. Adding more OLTP sessions drastically increased query execution time as well.

For HyPer, we use a transaction mix of 5 OLTP sessions and 3 parallel OLAP sessions executing queries. We do not make the simplification of running single-partition transactions only, as for VoltDB, but challenge HyPer with warehouse-crossing transactions as specified in Section 3.1. In one setup, the OLAP sessions operate on the initially loaded data (cf. Figure 9). In a second one, a fresh snapshot is created for every new query stream (cf. Figure 9). Queries are snapshot-isolated from transactions. On the OLTP side, the isolation level is serializable.

Since HyPer does not feature separate client and server processes yet, the results are produced by a single driver that incorporates both components. Thus, potential performance loss caused by inter-process communication is ruled out for HyPer, but not for the others systems under test. HyPer's strong OLTP performance results from the compilation of transactions to machine code. VoltDB uses stored procedures written in Java instead.

Figure 9 shows the memory consumption of HyPer and VoltDB. We do not include MonetDB results here, because the MonetDB database does not grow over time. HyPer runs three query sessions concurrently to the 5 OLTP sessions and spawns a fresh VM snapshot after each iteration. VoltDB executes a pure-OLTP workload. The figure shows the memory consumption after the initial load was performed.

Query	**System X** 3 query streams 25 JDBC clients OLTP throughput	**System X** 3 query streams 25 JDBC clients Query resp. times (ms)	**HyPer** configurations 8 query sessions (streams) single threaded OLTP OLTP throughput	**HyPer** configurations 8 query sessions (streams) single threaded OLTP Query resp. times (ms)	**HyPer** configurations 3 query sessions (streams) 5 OLTP threads OLTP throughput	**HyPer** configurations 3 query sessions (streams) 5 OLTP threads Query resp. times (ms)	**MonetDB** no OLTP 3 query streams Query resp. times (ms)	**VoltDB** no OLAP only OLTP
Q1	new order: 221.91 tps; total: **493.14** tps	4221	new order: 25166 tps; total: **55924** tps	76	new order: 112217 tps; total: **249237** tps	70	72	new order: 16273.80 tps; total: **36159.07** tps According to [Volb]: **53000** tps (total) on one node; 560000 tps (total) on 12 nodes
Q2		6555		282		156	218	
Q3		16410		112		72	112	
Q4		3830		348		227	8168	
Q5		15212		2489		1871	12028	
Q6		3895		24		15	163	
Q7		8285		2622		1559	2400	
Q8		1655		563		614	306	
Q9		3520		457		241	214	
Q10		15309		4288		2408	9239	
Q11		6006		48		32	42	
Q12		5689		324		182	214	
Q13		918		403		243	521	
Q14		6096		420		174	919	
Q15		6768		1407		822	587	
Q16		6088		2157		1523	7703	
Q17		5195		187		174	335	
Q18		14530		240		123	2917	
Q19		4417		292		134	4049	
Q20		3751		313		144	937	
Q21		9382		48		47	332	
Q22		8821		10		9	167	
Throughput		0.38 queries/s		10.49 queries/s		5.96 queries/s	1.21 queries/s	

Figure 8: Performance Comparison: System X, HyPer OLTP&OLAP, MonetDB (OLAP only), VoltDB (OLTP only)

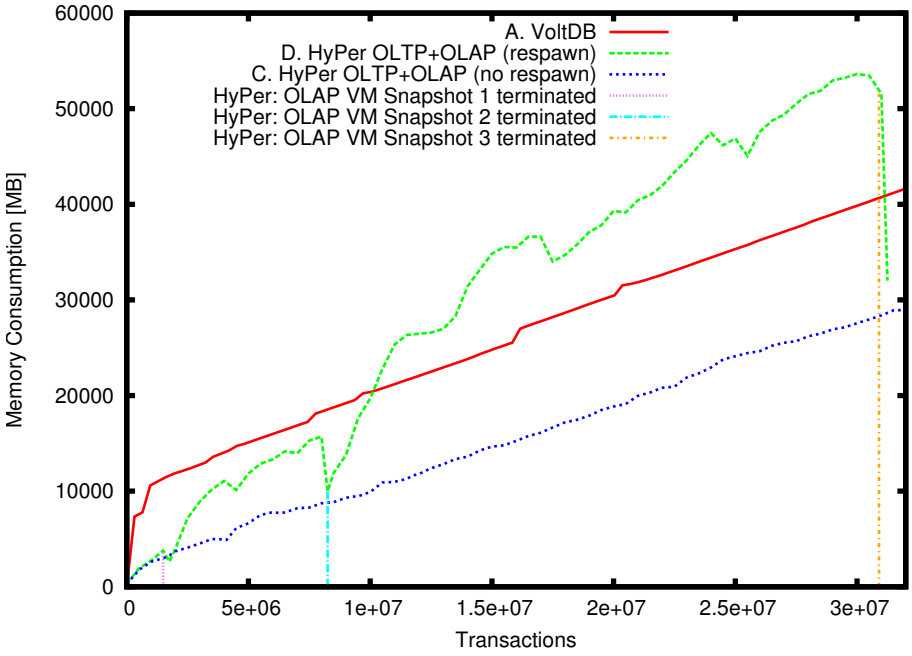


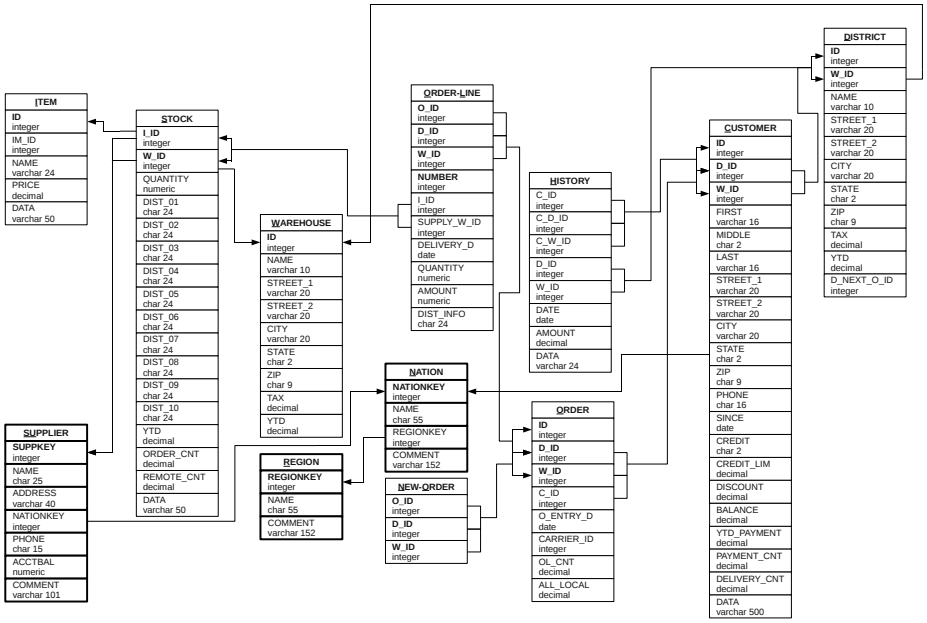
Figure 9: Memory consumption (after load) of VoltDB and HyPer

6 Summary

We presented TPC-CH, a benchmark for hybrid OLTP&OLAP database systems. TPC-CH is based on the standardized TPC-C and TPC-H benchmarks. It is not only suited for hybrid DBMS, but also allows to compare them with systems specialized on either of the two workloads as well as traditional, universal systems. We substantiate this claim with results of representatives of each type of database system.

Acknowledgment We acknowledge the fruitful discussions about this benchmark during the Dagstuhl Workshop on "Robust Query Processing" (September 2010). Stefan Krompaß helped to implement the System X benchmark.

A Schema



B Queries

All dates, strings and ranges in the queries are examples only. Due to space constraints, we had to relinquish pretty-printing, but include the queries formatted suitable for copy&paste.

Q1: Generate orderline overview

```
SELECT ol_number,SUM(ol_quantity) AS sum_qty,SUM(ol_amount) AS
sum_amount,AVG(ol_quantity) AS avg_qty,AVG(ol_amount) AS
avg_amount,COUNT(*) AS count_order FROM orderline WHERE
ol_delivery_d>'2007-01-02 00:00:00.000000' GROUP BY ol_number
ORDER BY ol_number
```

Q2: Most important supplier/item-combinations (those that have the lowest stock level for certain parts in a certain region)

```
SELECT su_suppkey,su_name,n_name,i_id,i_name,su_address,su_phone,
su_comment FROM item,supplier,stock,nation,region,(SELECT s_i_id
AS m_i_id,MIN(s_quantity) AS m_s_quantity FROM stock,supplier,
nation,region  WHERE mod((s_w_id*s_i_id),10000)=su_suppkey AND
su_nationkey=n_nationkey AND n_regionkey=r_regionkey AND r_name
LIKE 'Europ%' GROUP BY s_i_id) m WHERE i_id=s_i_id AND mod((s_w_id
```

```
 * s_i_id),10000)=su_suppkey AND su_nationkey=n_nationkey AND
n_regionkey=r_regionkey AND i_data LIKE '%b' AND r_name LIKE '
Europ%' AND i_id=m_i_id AND s_quantity=m_s_quantity ORDER BY
n_name,su_name,i_id
```

Q3: Unshipped orders with highest value for customers within a certain state

```
SELECT ol_o_id,ol_w_id,ol_d_id,SUM(ol_amount) AS revenue,o_entry_d
 FROM customer,neworder,"order",orderline WHERE c_state LIKE 'A%'
AND c_id=o_c_id AND c_w_id=o_w_id AND c_d_id=o_d_id AND no_w_id=
o_w_id AND no_d_id=o_d_id AND no_o_id=o_id AND ol_w_id=o_w_id AND
ol_d_id=o_d_id AND ol_o_id=o_id AND o_entry_d>'2007-01-02
00:00:00.000000' GROUP BY ol_o_id,ol_w_id,ol_d_id,o_entry_d ORDER
BY revenue DESC,o_entry_d
```

Q4: Orders that were partially shipped late

```
SELECT o_ol_cnt,COUNT(*) AS order_count FROM "order" WHERE
o_entry_d>='2007-01-02 00:00:00.000000' AND o_entry_d<'2012-01-02
00:00:00.000000' AND EXISTS (SELECT * FROM orderline WHERE o_id=
ol_o_id AND o_w_id=ol_w_id AND o_d_id=ol_d_id AND ol_delivery_d>=
o_entry_d) GROUP BY o_ol_cnt ORDER BY o_ol_cnt
```

Q5: Revenue volume achieved through local suppliers

```
SELECT n_name,SUM(ol_amount) AS revenue FROM customer,"order",
orderline,stock,supplier,nation,region WHERE c_id=o_c_id AND
c_w_id=o_w_id AND c_d_id=o_d_id AND ol_o_id=o_id AND ol_w_id=
o_w_id AND ol_d_id=o_d_id AND ol_w_id=s_w_id AND ol_i_id=s_i_id
AND mod((s_w_id * s_i_id),10000)=su_suppkey AND ascii(SUBSTRING(
c_state,1,1))=su_nationkey AND su_nationkey=n_nationkey AND
n_regionkey=r_regionkey AND r_name='Europe
                                                ' AND o_entry_d>='
2007-01-02 00:00:00.000000' GROUP BY n_name ORDER BY revenue DESC
```

Q6: Revenue generated by orderlines of a certain quantity

```
SELECT SUM(ol_amount) AS revenue FROM orderline WHERE
ol_delivery_d>='1999-01-01 00:00:00.000000' AND ol_delivery_d<'
2020-01-01 00:00:00.000000' AND ol_quantity BETWEEN 1 AND 100000
```

Q7: Bi-directional trade volume between two nations

```
SELECT su_nationkey AS supp_nation,cust_nation,o_year,SUM(
ol_amount) AS revenue FROM supplier,stock,orderline,(SELECT o_w_id
,o_d_id,o_id,o_c_id,EXTRACT(YEAR FROM o_entry_d) AS o_year FROM "
order") o,(SELECT c_id,c_w_id,c_d_id,c_state,SUBSTRING(c_state
,1,1) AS cust_nation FROM customer) c,nation n1,(SELECT
n_nationkey,n_name,code(n_nationkey) AS n_nationkeyasc FROM nation
) n2 WHERE ol_supplier_w_id=s_w_id AND ol_i_id=s_i_id AND mod((
s_w_id * s_i_id),10000)=su_suppkey AND ol_w_id=o_w_id AND ol_d_id=
o_d_id AND ol_o_id=o_id AND c_id=o_c_id AND c_w_id=o_w_id AND
```

```
c_d_id=o_d_id AND su_nationkey=n1.n_nationkey AND cust_nation=n2.
n_nationkeyasc AND ((n1.n_name='Germany                    ' AND n2.
n_name='Cambodia                  ')      OR (n1.n_name='Cambodia
                ' AND n2.n_name='Germany                   ')) AND
ol_delivery_d BETWEEN TIMESTAMP '2000-01-01' AND TIMESTAMP '
2099-01-01' GROUP BY su_nationkey,cust_nation,o_year ORDER BY
su_nationkey,cust_nation,o_year
```

Q8: Market share of a given nation for customers of a given region for a given part type

```
SELECT EXTRACT (YEAR FROM o_entry_d) AS l_year,SUM(CASE WHEN n2.
n_name='Germany                    ' THEN ol_amount ELSE 0 END)/SUM(
ol_amount) AS mkt_share FROM item,supplier,stock,orderline,"order"
,customer,nation n1,nation n2,region WHERE i_id=s_i_id AND ol_i_id
=s_i_id AND ol_supplier_w_id=s_w_id AND mod((s_w_id * s_i_id)
,10000)=su_suppkey AND ol_w_id=o_w_id AND ol_d_id=o_d_id AND
ol_o_id=o_id AND c_id=o_c_id AND c_w_id=o_w_id AND c_d_id=o_d_id
AND n1.n_nationkey=ascii(SUBSTRING(c_state,1,1)) AND n1.
n_regionkey=r_regionkey AND ol_i_id<1000 AND r_name='Europe
                                                    ' AND su_nationkey
=n2.n_nationkey AND o_entry_d BETWEEN '2007-01-02 00:00:00.000000'
 AND '2012-01-02 00:00:00.000000' AND i_data LIKE '%b' AND i_id=
ol_i_id GROUP BY l_year ORDER BY l_year
```

Q9: Profit made on a given line of parts,broken out by supplier nation and year

```
SELECT n_name,EXTRACT(YEAR FROM o_entry_d) AS l_year,SUM(ol_amount
) AS sum_profit FROM item,stock,supplier,orderline,"order",nation
WHERE ol_i_id=s_i_id AND ol_supplier_w_id=s_w_id AND mod((s_w_id *
 s_i_id),10000)=su_suppkey AND ol_w_id=o_w_id AND ol_d_id=o_d_id
AND ol_o_id=o_id AND ol_i_id=i_id AND su_nationkey=n_nationkey AND
 i_data LIKE '%BB' GROUP BY n_name,l_year ORDER BY n_name,l_year
DESC
```

Q10: Customers who received their ordered products late

```
SELECT c_id,c_last,SUM(ol_amount) AS revenue,c_city,c_phone,n_name
 FROM customer,"order",orderline,nation WHERE c_id=o_c_id AND
c_w_id=o_w_id AND c_d_id=o_d_id AND n_nationkey=ascii(SUBSTRING(
c_state,1,1)) AND ol_w_id=o_w_id AND ol_d_id=o_d_id AND ol_o_id=
o_id AND o_entry_d>='2007-01-02 00:00:00.000000' AND o_entry_d<=
ol_delivery_d GROUP BY c_id,c_last,c_city,c_phone,n_name ORDER BY
revenue DESC
```

Q11: Most important (high order count compared to the sum of all ordercounts) parts supplied by suppiers of a particular nation

```
SELECT s_i_id,SUM(s_quantity) AS ordercount FROM stock,supplier,
nation WHERE mod((s_w_id * s_i_id),10000)=su_suppkey AND
su_nationkey=n_nationkey AND n_name='Germany                    '
GROUP BY s_i_id HAVING SUM(s_quantity)>(SELECT SUM(s_quantity) *
.0001 FROM stock,supplier,nation WHERE mod((s_w_id * s_i_id)
```

```
,10000)=su_suppkey AND su_nationkey=n_nationkey AND n_name='
Germany                    ') ORDER BY ordercount DESC
```

Q12: Determine whether selecting less expensive modes of shipping is negatively affecting the critical-priority orders by causing more parts to be received late by customers

```
SELECT o_ol_cnt,SUM(CASE WHEN o_carrier_id=1 OR o_carrier_id=2
THEN 1 ELSE 0 END) AS high_line_count,SUM(CASE WHEN o_carrier_id
<>1 AND o_carrier_id<>2 THEN 1 ELSE 0 END) AS low_line_count FROM
"order",orderline WHERE ol_w_id=o_w_id AND ol_d_id=o_d_id AND
ol_o_id=o_id AND o_entry_d<= ol_delivery_d AND ol_delivery_d<'
2020-01-01 00:00:00.000000' GROUP BY o_ol_cnt ORDER BY o_ol_cnt
```

Q13: Relationships between customers and the size of their orders

```
SELECT c_count,COUNT(*) AS custdist FROM (SELECT c_id,COUNT(o_id)
FROM customer LEFT OUTER JOIN "order" ON (c_w_id=o_w_id AND c_d_id
=o_d_id AND c_id=o_c_id AND o_carrier_id>8) GROUP BY c_id) AS
c_orders (c_id,c_count) GROUP BY c_count ORDER BY custdist DESC,
c_count DESC
```

Q14: Market response to a promotion campaign

```
SELECT 100.00 * SUM(CASE WHEN i_data LIKE 'PR%' THEN ol_amount
ELSE 0 END) / 1+SUM(ol_amount) AS promo_revenue FROM orderline,
item WHERE ol_i_id=i_id AND ol_delivery_d>='2007-01-02
00:00:00.000000' AND ol_delivery_d<'2020-01-02 00:00:00.000000'
```

Q15: Determines the top supplier

```
with revenue (supplier_no,total_revenue) AS (SELECT supplier_no,
SUM(ol_amount) FROM orderline,(SELECT s_w_id,s_i_id,mod((s_w_id *
s_i_id),10000) AS supplier_no FROM stock) s WHERE ol_i_id=s_i_id
AND ol_supplier_w_id=s_w_id AND ol_delivery_d>='2010-05-23
12:00:00' GROUP BY supplier_no) SELECT su_suppkey,su_name,
su_address,su_phone,total_revenue FROM supplier,revenue WHERE
su_suppkey=supplier_no AND total_revenue=(SELECT MAX(total_revenue
) FROM revenue) ORDER BY su_suppkey
```

Q16: Number of suppliers that can supply parts with given attributes

```
SELECT i_name,brand,i_price,COUNT(DISTINCT (mod((s_w_id * s_i_id)
,10000))) AS supplier_cnt FROM stock,(SELECT i_id,i_data,i_name,
SUBSTRING(i_data,1,3) AS brand,i_price FROM item) i WHERE i_id=
s_i_id AND i_data NOT LIKE 'zz%' AND (mod((s_w_id * s_i_id),10000)
) NOT IN (SELECT su_suppkey FROM supplier WHERE su_comment LIKE '%
bad%') GROUP BY i_name,brand,i_price ORDER BY supplier_cnt DESC
```

Q17: Average yearly revenue that would be lost if orders were no longer filled for small quantities of certain parts

```
SELECT SUM(ol_amount) / 2.0 AS avg_yearly FROM orderline,item
WHERE ol_i_id=i_id AND i_data LIKE '%b' AND ol_quantity<(SELECT
0.2 * AVG(ol_quantity) FROM orderline WHERE ol_i_id=i_id)
```

Q18: Rank customers based on their placement of a large quantity order

```
SELECT c_last,c_id,o_id,o_entry_d,o_ol_cnt,SUM(ol_amount) FROM
customer,"order",orderline WHERE c_id=o_c_id AND c_w_id=o_w_id AND
 c_d_id=o_d_id AND ol_w_id=o_w_id AND ol_d_id=o_d_id AND ol_o_id=
o_id GROUP BY o_id,o_w_id,o_d_id,c_id,c_last,o_entry_d,o_ol_cnt
HAVING SUM(ol_amount)>200 ORDER BY SUM(ol_amount) DESC,o_entry_d
```

Q19: Machine generated data mining (revenue report for disjunctive predicate)

```
SELECT SUM(ol_amount) AS revenue FROM orderline,item WHERE (
ol_i_id=i_id AND i_data LIKE '%a' AND ol_quantity>=1 AND
ol_quantity<= 10 AND i_price BETWEEN 1 AND 400000 AND ol_w_id IN
(1,2,3)) OR (ol_i_id=i_id AND i_data LIKE '%b' AND ol_quantity>=1
AND ol_quantity<= 10 AND i_price BETWEEN 1 AND 400000 AND ol_w_id
IN (1,2,4)) OR (ol_i_id=i_id AND i_data LIKE '%c' AND ol_quantity
>=1 AND ol_quantity<= 10 AND i_price BETWEEN 1 AND 400000 AND
ol_w_id IN (1,5,3))
```

Q20: Suppliers in a particular nation having selected parts that may be candidates for a promotional offer

```
SELECT su_name,su_address FROM supplier,nation WHERE su_suppkey IN
 (SELECT mod(s_i_id * s_w_id,10000) FROM stock,orderline WHERE
s_i_id IN (SELECT i_id FROM item WHERE i_data LIKE 'co%') AND
ol_i_id=s_i_id AND ol_delivery_d>'2010-05-23 12:00:00' GROUP BY
s_i_id,s_w_id,s_quantity HAVING 2*s_quantity>SUM(ol_quantity)) AND
 su_nationkey=n_nationkey AND n_name='Germany                    '
ORDER BY su_name
```

Q21: Suppliers who were not able to ship required parts in a timely manner

```
Q21: SELECT su_name,COUNT(*) AS numwait FROM supplier,orderline l1
,"order",stock,nation WHERE ol_o_id=o_id AND ol_w_id=o_w_id AND
ol_d_id=o_d_id AND ol_w_id=s_w_id AND ol_i_id=s_i_id AND mod((
s_w_id * s_i_id),10000)=su_suppkey AND l1.ol_delivery_d>o_entry_d
AND NOT EXISTS (SELECT * FROM orderline l2 WHERE l2.ol_o_id=l1.
ol_o_id AND l2.ol_w_id=l1.ol_w_id AND l2.ol_d_id=l1.ol_d_id AND l2
.ol_delivery_d>l1.ol_delivery_d) AND su_nationkey=n_nationkey AND
n_name='Germany                    ' GROUP BY su_name ORDER BY
numwait DESC,su_name
```

Q22: Geographies with customers who may be likely to make a purchase

```
SELECT country,COUNT(*) AS numcust,SUM(c.c_balance) AS totacctbal
FROM (SELECT c_phone,c_balance,c_id,c_w_id,c_d_id,c_balance,
SUBSTRING(c_state,1,1) AS country FROM customer) c WHERE SUBSTRING
(c_phone,1,1) IN ('1','2','3','4','5','6','7') AND c.c_balance>(
SELECT AVG(c2.c_balance) FROM customer c2 WHERE c2.c_balance>0.00
AND SUBSTRING(c2.c_phone,1,1) IN ('1','2','3','4','5','6','7'))
AND NOT EXISTS (SELECT * FROM "order" WHERE o_c_id=c_id AND o_w_id
=c_w_id AND o_d_id=c_d_id) GROUP BY country ORDER BY country
```

C Relations

NATION		
NATION-KEY	NAME	REGION-KEY
48	Australia	4
49	Belgium	5
50	Cameroon	1
51	Denmark	5
52	Ecuador	2
53	France	5
54	Germany	5
55	Hungary	5
56	Italy	5
57	Japan	3
65	Kenya	1
66	Lithuania	5
67	Mexico	2
68	Netherlands	5
69	Oman	1
70	Portugal	5
71	Qatar	1
72	Rwanda	1
73	Serbia	5
74	Togo	1
75	United States	2
76	Vietnam	3
77	Singapore	3
78	Cambodia	3
79	Yemen	1
80	Zimbabwe	1
81	Argentina	2
82	Bolivia	2
83	Canada	2
84	Dominican Republic	2
85	Egypt	1
86	Finnland	5
87	Ghana	1
88	Haiti	2
89	India	3
90	Jamaica	4
97	Kazakhstan	3
98	Luxembourg	5
99	Morocco	1
100	Norway	5
101	Poland	5
102	Peru	2
103	Nicaragua	2
104	Romania	5
105	South Africa	1
106	Thailand	3
107	United Kingdom	5
108	Venezuela	2
109	Liechtenstein	5
110	Austria	5
111	Laos	3
112	Zambia	1
113	Switzerland	5
114	China	3
115	Papua New Guinea	4
116	East Timor	4
117	Bulgaria	5
118	Brazil	2
119	Albania	5
120	Andorra	5
121	Belize	2
122	Botswana	1

REGION		
REGIONKEY	NAME	...
1	Africa	
2	America	
3	Asia	
4	Australia	
5	Europe	

References

[BHF09] Carsten Binnig, Stefan Hildenbrand, and Franz Färber. Dictionary-based order-preserving string compression for main memory column stores. *International Conference on Management of Data*, 2009.

[BKS08] Anja Bog, Jens Krüger, and Jan Schaffner. A Composite Benchmark for Online Transaction Processing and Operational Reporting. *2008 IEEE Symposium on Advanced Management of Information for Globalized Enterprises (AMIGE)*, September 2008.

[BMK09] Peter A. Boncz, Stefan Manegold, and Martin L. Kersten. Database Architecture Evolution: Mammals Flourished long before Dinosaurs became Extinct. *PVLDB*, 2(2), 2009.

[CS04] Sang K Cha and Changbin Song. P * TIME : Highly Scalable OLTP DBMS for Managing Update-Intensive Stream Workload. *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, 2004.

[DHKK97] Jochen Doppelhammer, Thomas Höppler, Alfons Kemper, and Donald Kossmann. Database performance in the real world: TPC-D and SAP R/3. *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, 1997.

[GUMG10] Georgios Giannikis, Philipp Unterbrunner, Jeremy Meyer, and G. Crescando. *SIGMOD '10: Proceedings of the 2010 international conference on Management of data*, 2010.

[HAMS08] Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden, and Michael Stonebraker. OLTP through the looking glass, and what we found there. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, 2008.

[KGT+10] Jens Krueger, Martin Grund, Christian Tinnefeld, Hasso Plattner, Alexander Zeier, and Franz Faerber. Optimizing Write Performance for Read Optimized Databases. In *Database Systems for Advanced Applications*. Springer, 2010.

[KKN+08] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alex Rasin, Stanley B. Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg, and Daniel J. Abadi. H-store: a high-performance, distributed main memory transaction processing system. *PVLDB*, 1(2), 2008.

[KN11] Alfons Kemper and Thomas Neumann. HyPer – A Hybrid OLTP&OLAP Main Memory Database System Based on Virtual Memory Snapshots. *ICDE 2011: IEEE International Conference on Data Engineering*, 2011.

[Pla09] Hasso Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *Proceedings of the 35th SIGMOD international conference on Management of data*. ACM, 2009.

[SBKZ08] Jan Schaffner, Anja Bog, Jens Krüger, and Alexander Zeier. A Hybrid Row-Column OLTP Database Architecture for Operational Reporting. *Business Intelligence for the Real Time Enterprise (BIRTE 2008)*, 2008.

[Tra10a] Transaction Processing Performance Council. TPC Benchmark C (Standard Specification), 2010.

[Tra10b] Transaction Processing Performance Council. TPC Benchmark H (Standard Specification), 2010.

[Tra10c] Transaction Processing Performance Council. TPC-C Results. `http://www.tpc.org/downloaded_result_files/tpcc_results.txt`, 2010.

[Tra10d] Transaction Processing Performance Council. TPC-E Results. `http://www.tpc.org/downloaded_result_files/tpce_results.txt`, 2010.

[Vola] VoltDB TPC-C-like Benchmark Comparison-Benchmark Description. `http://community.voltdb.com/node/134`.

[Volb] VoltDB Inc. VoltDB: Product Overview.