ScyPer: Elastic OLAP Throughput on Transactional Data*

Tobias Mühlbauer
Technische Universität München
Munich, Germany
muehlbau@in.tum.de

Wolf Rödiger
Technische Universität München
Munich, Germany
roediger@in.tum.de

Angelika Reiser
Technische Universität München
Munich, Germany
reiser@in.tum.de

Alfons Kemper
Technische Universität München
Munich, Germany
kemper@in.tum.de

Thomas Neumann
Technische Universität München
Munich, Germany
neumann@in.tum.de

ABSTRACT

Ever increasing main memory sizes and the advent of multi-core parallel processing have fostered the development of in-core databases. Even the transactional data of large enterprises can be retained in-memory on a single server. Modern in-core databases like our HyPer system achieve best-of-breed OLTP throughput that is sufficient for the lion's share of applications. Remaining server resources are used for OLAP query processing on the latest transactional data, i.e., real-time business analytics. While OLTP performance of a single server is sufficient, an increasing demand for OLAP throughput can only be satisfied economically by a scale-out.

In this work we present ScyPer, a *Sc*ale-out of our H*yPer* main memory database system that horizontally scales out on shared-nothing hardware. With ScyPer we aim at (i) sustaining the superior OLTP throughput of a single HyPer server, and (ii) providing elastic OLAP throughput by provisioning additional servers on-demand, e.g., in the Cloud.

1. INTRODUCTION

Declining DRAM prices have lead to ever increasing main memory sizes. Together with the advent of multi-core parallel processing, these two trends have fostered the development of in-core database systems, i.e., systems that store and process data solely in main memory. On the high end, Oracle recently announced the SPARC M5-32 [6] with up to 32 CPUs and 32 TB of main memory in a single machine. While the M5-32 certainly has a high price tag, servers with 1 TB of main memory are already retailing for less than $35,000. On such a server, in-core databases like our HyPer system [3] process more than 100,000 TPC-C transactions (TX) per second in a single thread, which is enough for human generated workloads even during peak hours. A ballpark estimate of Amazon's yearly transactional data volume further reveals that retaining all data in-memory is feasible even for large enterprises: with a revenue of $60 billion, an average item price of $15, and about 54 B per orderline, we derive less than 1/4 TB for the orderlines—the dominant repository in a sales application. We thus conjecture that for the lion's share of OLTP workloads, a single server suffices.

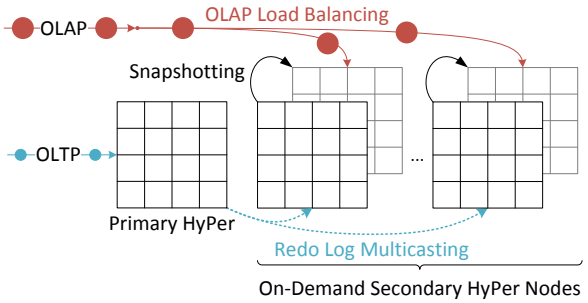


Figure 1: Elastic provisioning of secondary HyPer nodes for scalable OLAP throughput

Besides OLTP, data management solutions are today also faced with analytical workloads (OLAP). It is common to run these analytical queries in a separate data warehouse to avoid interference with the mission-critical OLTP processing. The data warehouse is updated only periodically (e.g., every night), which inevitably leads to the problem of data staleness. Industry leaders like SAP's Hasso Plattner [9] argue that this does not suit today's business needs and call for a real time business analytics paradigm, which aims at the analysis of fresh transactional data. Emerging hybrid main memory databases like SAP's HANA or HyPer address this issue. HyPer achieves best-of-breed OLTP throughput and OLAP query response times in one system in parallel on the same database state. Even though available resources—i.e., CPU cores that are not used for OLTP—can process OLAP queries, OLAP throughput is still limited.

In this work we present ScyPer, a *Sc*aled-out version of our H*yPer* main memory database system that horizontally scales out on shared-nothing hardware, e.g., in the Cloud. With ScyPer we aim at (i) sustaining the superior OLTP throughput of a single HyPer server, and (ii) providing elastic OLAP throughput by provisioning additional servers on-demand. Fig. 1 gives an overview of its architecture.

*Tobias Mühlbauer is a recipient of the Google Europe Fellowship in Structured Data Analysis, and this research is supported in part by this Google Fellowship. Wolf Rödiger is a recipient of the Oracle External Research Fellowship. This work has been sponsored by the German Federal Ministry of Education and Research (BMBF) grant HDBC 01IS12026.

DanaC'13, June 23, 2013, New York, NY, USA

In ScyPer, a primary node processes incoming TX and multicasts the redo log to secondaries, which in turn replay the log and—mainly due to not having to replay reader and aborted TX—have free capacities to process OLAP queries on TX-consistent virtual memory snapshots. HyPer's efficient snapshotting mechanism, which is based on the `fork()` system call, is one of the cornerstones of its superior OLTP and OLAP performance. Secondaries can further use free hardware resources to maintain additional indexes, access non-transactional data, and write out database backups.

In particular, this work makes the following contributions:

- We show *how the redo log is propagated* from the primary server to secondaries using a reliable multicasting protocol. An evaluation of this approach on a 1 GbE and a 40 Gbit/s InfiniBand (4×QDR IPoIB) infrastructure demonstrates the feasibility of this approach. Further, we compare logical and physical redo logging in the context of ScyPer.
- We describe how *global TX-consistent snapshots* are created in ScyPer and which guarantees they give.
- We evaluate the *sustained OLTP and scalable OLAP throughput* of ScyPer using the TPC-CH [1] benchmark that combines the transactional TPC-C and analytical TPC-H workloads.
- We show that secondaries can act as *high availability failovers* for the primary HyPer instance.

ScyPer in the Cloud. "In-memory computing will play an ever-increasing role in Cloud Computing" [9]: ScyPer with its elastic scale-out is particularly suitable for the deployment on Cloud infrastructures. In an infrastructure-as-a-service scenario, ScyPer runs on multiple physical machines in the Cloud. Nodes for secondaries are rented on-demand, which makes this model highly cost-effective. In a database-as-a-service scenario, ScyPer is offered as a service. The service provider aims at an optimal resource usage. Following the partitioned execution model of H-Store [2] and VoltDB, HyPer—and thus primary ScyPer instances—provides high single-server OLTP throughput on multiple partitions in parallel, which allows running multiple tenants on one physical machine [5]. Redo logs for the partitions are multicast on a per-tenant basis so that OLAP secondaries can be created for specific tenants. OLAP processing of multiple tenants can again be consolidated on a single server. In case a primary node that processes the OLTP of multiple tenants faces an increased load, partitions of a tenant can migrate from one primary to another or a secondary can take over as a primary very quickly (see Sect. 2.4). In summary, ScyPer in the Cloud allows great flexibility, very good resource utilization, and high cost-effectiveness. However, let us add a word of caution: many Cloud infrastructure offerings are virtualized. Our previous experiments, which are out of scope for this work, suggest that running applications tuned for modern hardware like ScyPer on such instances can lead to severe performance degradations; unvirtualized instances should thus be preferred.

2. SCYPER

ScyPer consists of two HyPer instance types: one primary and multiple secondaries. Incoming OLTP is processed on the primary while OLAP queries are load-balanced across secondaries (and the primary if it has spare resources). This allows to scale the OLAP throughput by provisioning additional secondaries on-demand. When a secondary instance is started, it first fetches the latest full database backup from durable storage and then replays the redo log until it catches up with the primary. Secondaries can always catch up as redo log replaying is about ×2 faster than processing the original OLTP workload (see Sect. 2.1). Further, the system will not experience full load at all times.

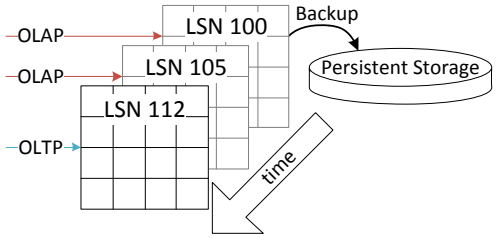


Figure 2: Long-running snapshots allow parallel processing of OLAP queries and simultaneous backups.

The primary node usually uses a row-store data layout which is better suited for OLTP processing and it keeps indexes that support efficient transaction (TX) processing. When processing the OLTP workload, the primary node multicasts the redo log of committed TX to a specific multicast address. The address encodes the database partition such that secondaries can subscribe to specific partitions. This allows the provisioning of secondaries for specific partitions and enables a more flexible multi-tenancy model as described in Sect. 1. Besides being multicast, the log is further sent to a durable log. Each redo log entry for a TX comes with a log sequence number (LSN). ScyPer uses these LSNs to define a logical time in the distributed setting. A secondary that last replayed the entry with LSN x has logical time x. It next replays the entry with LSN $x + 1$ and advances its logical time to $x + 1$.

As a large portion of a usual OLTP workload is read-only (i.e., no redo is necessary), replaying the redo log on secondary nodes is usually cheaper than processing the original workload on the primary node. Further, read operations of writer TX do not need to be evaluated when physical logging is employed. The available resources on the secondaries are used to process incoming OLAP queries on TX-consistent snapshots. HyPer's efficient snapshotting mechanism allows to process several OLAP queries in parallel on multiple snapshots as shown in Fig. 2. A snapshot can also be written to persistent storage so that it can be used as a TX-consistent starting point for recovery. Furthermore, the faster OLTP processing allows to create additional indexes for efficient analytical query processing. Secondary nodes can either store data in a row-, column-, or a hybrid row- and column-store data format. Additionally, these nodes can include non-transactional data in OLAP analyses which need not necessarily be kept in-core.

In the following we describe our redo log propagation and distributed snapshotting approaches. We further show how ScyPer provides scalable OLAP throughput while sustaining the OLTP throughput of a single server and how secondary nodes can act as high availability failovers.

2.1 Redo Log Propagation

When processing a TX, HyPer creates a memory-resident undo log which is used for the rollback of aborted TX. Additionally, a redo log is created. For committed TX, this

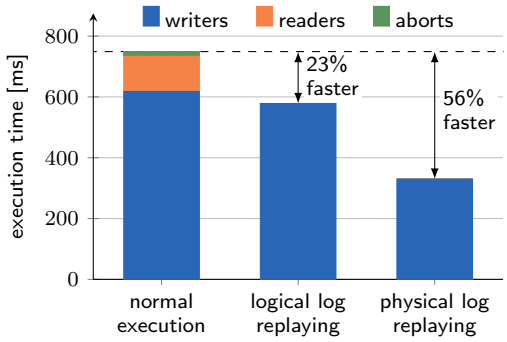


Figure 3: Time savings when replaying 100k TPC-C transactions using logical and physical redo logging

redo log has to be persisted and written to durable storage so that it can be replayed. The undo log however can be discarded when a TX commits.

ScyPer uses multicasting to propagate the redo log of committed TX from the primary to secondary nodes to keep them up-to-date with the most recent transactional state. Multicasting allows to add secondaries on-demand without increasing the network bandwidth usage.

UDP vs. PGM multicast. Standard UDP multicasting is not a feasible solution for ScyPer as it may drop messages, deliver them multiple times, or transfer them out of order. Instead, ScyPer uses OpenPGM for multicasting, an open source implementation of the Pragmatic General Multicast (PGM) protocol [8], which is designed for reliable and ordered transmission of messages from a single source to multiple receivers. Receivers detect message loss and recover by requesting a retransmission from the sender.

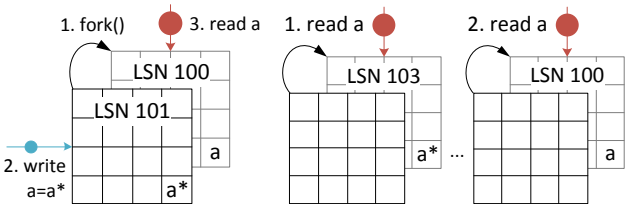
Logical vs. physical logging. ScyPer supports both, the use of logical and physical redo logs for redo log propagation. These two alternatives differ in the size of the resulting log and the time needed to replay it. While in a logical redo log only the TX identifier and invocation parameters are logged, the physical redo log logs the individual insert, update, and delete statements that modified the database during the TX. Physical redo logging results in a larger log but replaying it is often much faster compared to logical logging, especially when the logged TX executed costly logic or many read operations. In any case, TX that use operations where the outcome can not be determined solely by the transactional state, e.g., random operations or current time information, have to be logged using physical redo logging. It is of note that logical redo logging is restricted to pre-canned stored procedures. However, stored procedures can be added to ScyPer at any time by a low-overhead system-internal TX.

As mentioned before, secondaries do not need to replay all TX. Only committed TX that modified data are logged. Fig. 3 shows that replaying the logical log of 100,000 TPC-C TX saves 17% in execution time compared to the original processing of the TX by not having to re-execute reader and aborted TX and an additional 6 % for not having to log again (undo and redo log)—together this adds up to savings of 23 %. Physical logging is even able to save 56 % of execution time as it further does not re-execute read operations of writers and only replays basic inserts, updates and deletes.

The physical log for 100,000 TPC-C TX has a size of 85 MB and is therefore about ×5 larger than the logical log which needs only 14 MB. An individual physical log entry has an average size of ∼1,500 B, whereas a logical log entry has ∼250 B. Group commits allow to bundle and compress log entries for improved network usage. Compression is not feasible on a per-TX basis as the individual log entries are simply too small. Compressing the log for 100,000 TPC-C transactions using LZ4 compression reduces the size by 48 % in the case of physical and by 54 % for logical logging.

	1 GbE		InfiniBand	
	UDP	PGM	UDP	PGM
Bandwidth [Mbit/s]	906	675	14,060	1,832
Throughput [1,000/s]	81	43	1,252	112
Latency [μs]	100.4		13.5	

Table 1: Comparison of UDP and PGM performance for Gigabit Ethernet and InfiniBand 4×QDR



(a) Local order violation (b) Diverging distributed reads

Figure 4: Two problems which lead to unexpected results prevented by global TX-consistent snapshots

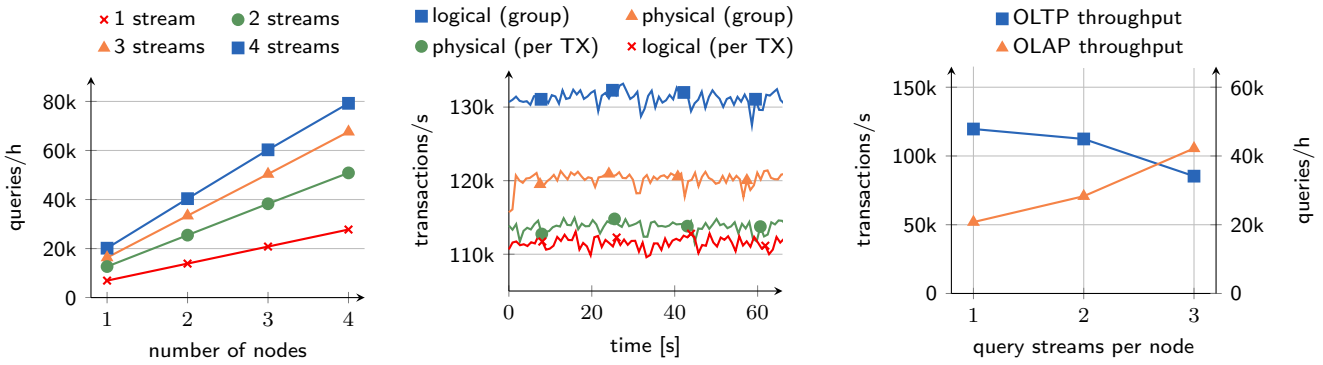
Ethernet vs. InfiniBand. Table 1 compares the single-threaded performance of UDP and PGM multicast in a 1 Gigabit Ethernet (1 GbE) and a 4×QDR IPoIB InfiniBand infrastructure. Our setup consists of four machines each equipped with an on-board Intel 82579V 1 GbE adapter and a Mellanox ConnectX-3 InfiniBand adapter (PCIe 3 ×8). We used a standard 1 GbE switch and a Mellanox 8 Port 40 Gbit/s QSFP switch. UDP was measured with 1.5 kB datagrams; PGM messages had a size of 2 kB. The UDP bandwidth and throughput increases by a factor of 15 from 1 GbE to InfiniBand; PGM still profits by a factor of 2.7. The latency is, in both cases, reduced by a factor of 7.

With a processing speed of around 110,000 TPC-C TX per second, HyPer creates ∼60,000 redo log entries per second per OLTP thread. 1 GbE allows the multicasting of the 60,000 logical log entries but offers not enough performance for physical logging due to its low PGM multicast performance. Only when group commits with log compression are used, physical redo log entries can be multicast over 1 GbE. Our InfiniBand setup can handle physical redo logging without compression and even has free capacities to support multiple outgoing multicast streams. These could be used for the simultaneous propagation of the redo logs of all TX-processing threads in a partitioned execution setting.

2.2 Distributed Snapshots

ScyPer adapts HyPer's efficient virtual memory snapshotting mechanism [3] to the distributed setting. In the following, we describe how we designed ScyPer's global TX-consistent snapshotting mechanism to solve two potential problems which affect query processing on transactional data: *local order violations* and *diverging distributed reads*.

Local order violations. Fig. 4(a) shows a schedule which exhibits a local order violation: First, the snapshot is cre-



(a) OLAP throughput on different numbers of nodes (without TX load)

(b) TX throughput on the primary with redo log propagation to secondaries

(c) Combined processing of OLTP and OLAP with different numbers of OLAP query streams

Figure 5: Evaluation of ScyPer's isolated and combined OLAP and OLTP throughput on a 4 node cluster

ated. Then a TX modifies a data item which is afterwards read by an OLAP query. In this example the query reads the data item's old value `a` because the snapshot was created before the TX changed it to `a*`. A single client who issued both, the TX and the query, would get an unexpected result—even though the schedule satisfies serializability. Order-preserving serializability (OPS) avoids such order violations as it "requires that transactions that do not overlap in time appear in the same order in a conflict-equivalent schedule" [10]. In the example the TX finished before the query, i.e., both did not overlap, therefore OPS requires that the query reads the new state.

To achieve OPS, a query has to be executed on a snapshot that is created after its arrival. While one might argue that if OPS is desired, the query has to be executed as a TX, we propose a solution that does not require this. A simple solution is to create a snapshot for every single query. However, while snapshot creation is cheap, it does not come for free. Therefore, we associate queries with a logical arrival time and delay their execution until a snapshot with a greater logical creation time is available. The primary node then acts as a load balancer for OLAP queries and tags every incoming query with a new LSN as its logical arrival time. The primary also triggers the periodic creation of global TX-consistent snapshots (e.g., every second). Together, this guarantees order-preserving serializability as TX are executed sequentially and queries always execute on a state of the data set that is newer than their arrival time.

Diverging distributed reads. The system as described until now is further subject to a problem which we call diverging reads: Executing the same OLAP query twice can lead to two different results in which the second result is based on an older transactional state—i.e., a database state with a smaller LSN than that of the first query. Fig. 4(b) shows an example for this. The diverging reads problem is caused by the load balancing mechanism, which may assign a successive query to a different node whose snapshot represents the state of the data set for an earlier point in time. This problem is not covered by order-preserving serializability but is solved by the synchronized creation of snapshots.

To create such a global snapshot, the primary node sends a system-internal TX to the secondary nodes which create local snapshots using the `fork()` system call at the logical time point defined by the transaction's LSN. We use a logical time based on LSNs to avoid problems with clock skew across nodes. The creation of the global TX-consistent snapshot is fully asynchronous on the primary node which avoids any interference with transaction processing. Therefore, the time needed to create a global TX-consistent snapshot only affects the OLAP response time on the secondaries. The time to create a global TX-consistent snapshot on n secondary nodes is defined by

$$\max_{0 \le i < n} (\mathrm{RTT}_i + T_{\mathrm{replay}_i} + T_{\mathrm{fork}_i})$$

where RTT_i is the round trip time from the primary to secondary i, T_{replay_i} is the time required to replay the outstanding log at i, and T_{fork_i} is the time to fork at i. In our high-speed InfiniBand setup, RTTs are as low as a few μs. To avoid inconsistencies, the snapshot transaction has to be processed in order, i.e., the outstanding log at the secondary has to be processed first. However, it is expected that at most one TX has to be replayed before the snapshot can be created—as the secondaries process TX faster than they arrive. On average a transaction takes only 10 μs. The time of the `fork` depends on the memory page size and the database size but is in general very fast, e.g., with a database size of 8 GB a `fork` takes 1.5 ms with huge and 50 ms with small pages. All in all, the time needed to create a global TX-consistent snapshot adds up to only a few milliseconds which has no significant impact on the OLAP response time.

Distributed processing. As global TX-consistent snapshots avoid inconsistencies between local snapshots, they also enable the distributed processing of a single query on multiple secondaries. The distributed processing has the potential to further reduce query response times.

2.3 Scaling OLAP Throughput on Demand

The evaluation of ScyPer was conducted on four commodity workstations, each equipped with an Intel Core i7-3770 CPU and 32 GB dual-channel DDR3-1600 DRAM. The CPU is based on the Ivy Bridge microarchitecture, has 4 cores, 8 hardware threads, a 3.4 GHz clock rate, and 8 MB of last-level shared L3 cache. As operating system we used Linux 3.5 in 64 bit mode. Sources were compiled using GCC 4.7 with `-O3 -march=native` optimizations.

Fig. 5 shows the isolated and combined TPC-CH benchmark [1] OLAP and OLTP throughput that can be achieved with the ScyPer system. We prioritized the OLTP process so that replaying the log is preferred over OLAP query process-

ing to avoid that secondaries cannot keep up with redo log replaying. Fig. 5(a) demonstrates that the OLAP throughput scales linearly when no TX are processed at the same time. Multiple query streams allow the nodes to process queries in parallel using their 4 cores and therefore increase the OLAP throughput. Fig. 5(b) shows the TX throughput which was achieved on the primary node while the redo log is simultaneously broadcasted to and replayed by the secondaries. The figure shows the TX rate for different redo log types and commit variants. While the TX rates for the four options differ by at most 15 %, group commits clearly provide a better performance than per-TX log propagation. The reason for this is the reduced PGM processing overhead since group commits lead to fewer and larger messages. Finally, Fig. 5(c) shows the combined execution of OLTP and OLAP with logical redo log propagation and uncompressed group commits. All four nodes, including the primary, process OLAP queries. The nodes are able to handle up to two query streams each, while sustaining a OLTP throughput of over 100,000 TX/s (normal execution on primary, replaying on secondaries). Three streams degrade the OLTP throughput noticeably and with four streams, secondaries can no longer keep up with TX log replaying. This is reasonable, as the nodes only have 4 cores, of which in this case all are busy processing queries.

2.4 High Availability

Besides providing scalable OLAP throughput on transactional data, secondary HyPer nodes can further be used as high availability failover nodes. Secondaries detect the failure of the primary when no redo log message—or heartbeat message—is received from the primary within a given timeframe. In case of failure, the secondaries then elect a new primary using a distributed consensus protocol. The new primary and the remaining secondaries replay all TX in the durable redo log for which they have not yet received the multicast log. Once the primary replayed these TX, it is active and can process new TX. It is thus recommendable to chose the new primary depending on the number of TX it has to replay, i.e., to choose the secondary with smallest difference between its LSN and the LSN of the durable redo log. Further, if a secondary using a row-store layout exists, this node should be preferred over nodes using a column-store layout. However, for our main memory database system HyPer, TPC-C TX processing performance only decreases by about 10 % using a column- compared to a row-store layout. In conclusion, ScyPer is designed to handle a failure of its primary node within a very short period of time.

3. RELATED WORK

Oracle's Change Data Capture (CDC) system [7] allows to continuously transfer updates of the OLTP database to the data warehouse. Instead of periodically running an extract-transform-load phase, CDC allows changes to be continually captured in the warehouse. As in ScyPer, changes can be sniffed from the redo log to minimize the load on the OLTP database. In contrast to CDC, ScyPer, however, multicasts the redo log directly from the primary to subscribed secondaries, which allows updates to be propagated with a μs latency in modern high-speed network infrastructures. Further, being an in-core database, ScyPer allows unprecedented transaction and query processing speeds that are unachieved by current commercial solutions. As ScyPer consolidates the transactional database and the data warehouse in a true hybrid OLTP&OLAP solution, secondaries can further act as failovers for the primary. This is comparable to Microsoft's SQL Server AlwaysOn solution [4], which allows multiple SQL Server instances to be running as backups that can take over quickly in case of a failure.

ScyPer differs from traditional data warehousing by not relying on materialized views, which are commonly used to speed up query processing. In-core processing of queries allows clock-speed scan performance, which in turn makes a high query throughput and superior response times possible.

4. CONCLUSION AND OUTLOOK

In this work we have shown that ScyPer, a scaled-out version of the HyPer in-core database system, is indeed able to sustain the superior OLTP throughput of a single HyPer server while providing elastic OLAP throughput by provisioning additional servers on-demand. OLAP queries are thereby executed on global TX-consistent snapshots of the transactional state. We have shown that ScyPer's snapshotting mechanism guarantees order-preserving serializability and further prevents the problem of diverging reads in a distributed setting. Secondary nodes are efficiently kept up-to-date using a redo log propagation mechanism based on reliable multicasting. In case of a primary node failure, these secondaries act as high availability failovers.

In the future we plan to extend ScyPer's query engine to handle distributed queries on fragmented transactional data sets. Further, our evaluation has revealed that careful resource management is crucial for high-performance hybrid query and transaction processing.

5. REFERENCES

[1] R. Cole et al. The mixed workload CH-benCHmark. In *DBTest*, 2011.

[2] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. Jones, et al. H-store: a high-performance, distributed main memory transaction processing system. *PVLDB*, 1(2), 2008.

[3] A. Kemper and T. Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *ICDE*, 2011.

[4] Microsoft. SQL Server: High Availability. `http://microsoft.com/en-us/sqlserver/solutions-technologies/mission-critical-operations/high-availability.aspx`.

[5] H. Mühe, A. Kemper, and T. Neumann. The mainframe strikes back: elastic multi-tenancy using main memory database systems on a many-core server. In *EDBT*, 2012.

[6] Oracle. SPARC M5-32 Server. `http://www.oracle.com/us/products/servers-storage/servers/sparc/oracle-sparc/m5-32/overview/index.html`.

[7] Oracle 11g: Change Data Capture. `http://docs.oracle.com/cd/B28359_01/server.111/b28313/cdc.htm`.

[8] PGM Reliable Transport Protocol Specification. `http://tools.ietf.org/pdf/rfc3208`.

[9] H. Plattner and A. Zeier. *In-Memory Data Management: An Inflection Point for Enterprise Applications*. Springer, 2011.

[10] G. Weikum and G. Vossen. *Transactional Information Systems*. Morgan Kaufmann, 2002.