The Mainframe Strikes Back: Multi Tenancy in the Main Memory Database HyPer on a TB-Server

Henrik Mühe, Alfons Kemper, Thomas Neumann

TU München
Boltzmannstr. 3
85748 Garching, Germany
muehe@in.tum.de kemper@in.tum.de neumann@in.tum.de

Abstract: Contrary to recent trends in database systems research focussing on scaling out workloads on a cluster of commodity computers, this presentation will break grounds for scale-up. We show that an elastic multi-tenancy solution can be achieved by combining a many-core server with a low footprint main memory database system. Total transactional throughput for TPC-C like order-entry transactions reaches up to 2 million transactions per second on a 32 core server while the number of tenants sharing a single server can be varied from a few to hundreds of separate tenants without diminishing total throughput. Contrary to common belief, a scale-up solution provides high flexibility for tenants with growing throughput needs and allows for simple sharing of common resources between different tenants while minimizing hardware and computing overhead. We show that our approach can handle changes in tenant requirements with minimal impact on other tenants on the server. Additionally, we prove that our architecture provides sufficient per-tenant throughput to handle big tenants and scales well with database size.

1 Introduction

Over the last years – perhaps fueled by observing the scale-out strategy of Google and other web service providers – the database community has largely focused on scale out using large numbers of low-cost commodity computers. An example of a provider offering this kind of hardware infrastructure is Amazon's cloud offering called EC2, which allows customers to obtain resources in the form of predefined machine specifications – just as if they rented a commodity computer hosted inside Amazon's data center[1]. In the area of database systems, cloud services typically offer two types of data stores: On the one hand, a key-value store implementation for data with low consistency requirements, like for instance click logs for a company's website. On the other hand, a regular relational database system hosted inside one of the predefined machines which can be rented by their customers. Unfortunately, the latter implementation of a relational database in the cloud is in stark contrast to the widespread idea of cloud computing. According to wikipedia,

[1]`http://aws.amazon.com/ec2/`

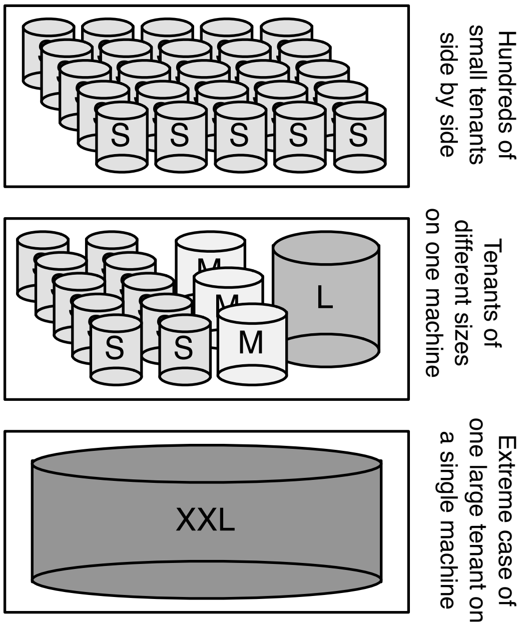
Hundreds of small tenants side by side
S
S
S
S
S
Tenants of different sizes on one machine
S
S
M
M
M
L
Extreme case of one large tenant on a single machine
XXL

Figure 1: A range of usage scenarios – no explicit reconfiguration is required to adapt the cloud database server.

cloud computing provides a service – in this case a relational data store – instead of a server hosting a DBMS. The latter idea of a database in the cloud leads to multiple drawbacks:

Scalability: Since computing resources specified by a fixed machine profile are rented and used as a database server, no additional flexibility is offered over an actual computer hosted in a data center. If migration to a different, more powerful machine profile is at all possible, the process is not seamless and requires substantial user involvement which can lead to user error[2].

Sharing of resources: With many applications, data can be shared between multiple tenants [ASJK11]. Using virtualization to share hardware between multiple tenants effectively prohibits this kind of resource sharing.

Overhead: Virtualization as well as the high footprint of traditional database systems incur substantial overhead. Without this additional layer between data and hardware, existing resources can be used more efficiently.

In this paper, we advocate combining multi-core server hardware with a low-footprint main memory database system to create an elastic and low-cost multi-tenancy setup. Commodity servers with 32 cores and main memory sizes of up to 1 Terabyte are widely available today for less than $35,000 (c.f. Figure 2). Combined with a state of the art transactional

[2]An example of a user error when managing cloud resources in this context is described in `http://bit.ly/nZcX84`

Figure 2: A commodity server capable of a combined TPC-C like transaction throughput of up to 2 million transactions per second. Notice the keyboard on top of the machine for scale.

main memory database system like VoltDB [Vol10] or our prototype database system HyPer [KN11], an overall throughput in the order of millions of transactions per second for TPC-C like order-entry transactions can be achieved.

Instead of designing complex mechanisms to separate different tenants in one database system, our approach relies on the operating system to map different tenants' databases to specific cores to achieve maximum throughput. This keeps the overhead of the database system to a minimum and allows for features like different scheduling strategies and prioritization implemented in the OS to be used without cost.

2 Technical Realization

We have developed the HyPer database system prototype [KN11] which reduces the memory footprint of traditional database systems to a minimum. Our prototype consumes less than 100kb of memory for an empty database with the TPC-C schema and an additional 10MB of memory for the statically linked executable. Despite the low memory overhead and small executable size, HyPer is capable of running OLTP transactions comparable to those found in the TPC-C at a rate of 100,000 transactions per seconds per thread. Additionally, HyPer is capable of executing long-running OLAP queries on an arbitrarily recent snapshot without severely impacting transaction throughput as we showed in [KN11] and demonstrated in [FKN11].

The high transaction throughput is achieved by reengineering a main memory database system from scratch, removing superfluous components formerly required in disk-based database systems and by using compilation over interpretation. Therefore, HyPer makes extensive use of the operating system's virtual memory management, removing the need for a buffer manager and executes transactions serially as pioneered by H-Store [KKN$^+$08], thus eliminating the need for a lock manager. To further increase performance, HyPer compiles pre-canned transactions as well as OLAP queries to machine code using the LLVM compiler infrastructure as illustrated in [Neu11]. All schema specific parts – like reading an attribute from a tuple or finding a tuple using an index – are generated as relation specific machine code, whereas general fragments like for instance a join operator are written in C++ and invoked from the generated code. Moreover, HyPer is fully ACID compliant, as atomicity and isolation directly follow from employing single threaded execution for transactions.

#Tenants	Average TPS	Overall TPS
1	80,729	81k
2	80,767	162k
4	81,060	324k
8	67,018	536k
16	58,605	938k
32	54,087	1731k
64	34,319	2197k
128	16,812	2152k
256	7,927	2029k
512	3,746	1919k

Table 1: Example configurations with a varying number of homogeneous tenants.

To support multiple tenants, multiple HyPer instances are launched, one for each tenant. Because of the compact footprint of the DBMS, even several hundred instances run on the server displayed in Figure 2 cause only a minimal memory overhead of less than 0.1%. When an instance does not execute any transactions or queries, no background tasks or batch processes are run, effectively reducing the CPU cycles required to run an idle instance to zero. Allocating tenants to a specific processor or migrating them to another one in case of a situation with unbalanced load is done by the operating system using the integrated scheduler. Resource sharing between different instances is achieved using shared memory segments mapped into the virtual memory of multiple instances. That way, read-only data can be shared with minimal implementation overhead. For data that needs to be updated infrequently, regular mutual exclusions as provided by the operating system ensure consistency in shared segments.

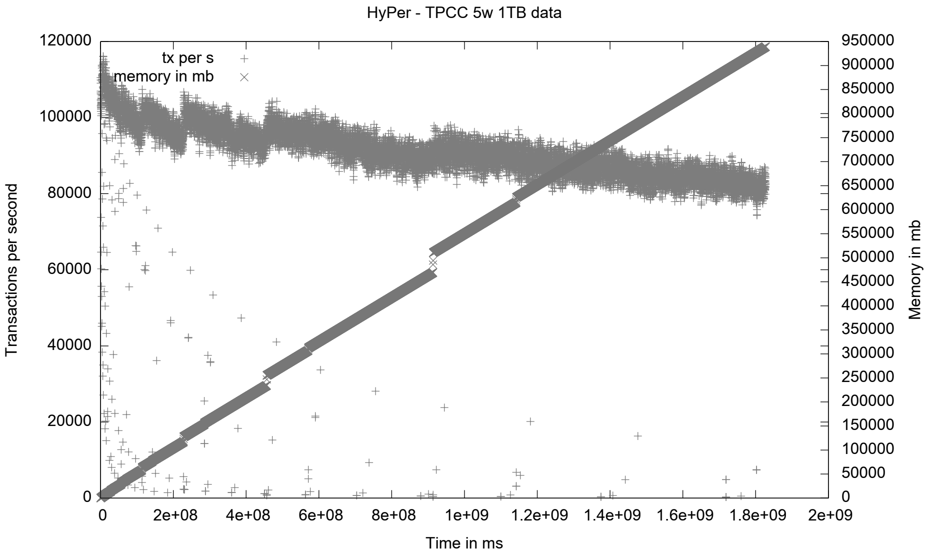

Figure 3: HyPer main memory database system growing form 500MB to 950GB of data with almost constant throughput.

Since resource distribution is dynamically controlled by the operating system, a tenant with growing throughput or memory demands will receive more resources without any explicit intervention. Table 1 shows average throughput and overall throughput for the same benchmark with a varying number of homogeneous clients. First, it can be observed that the overall throughput reaches a peak at 64 tenants which is the number of available hardware threads. Second, the sustained total throughput does not significantly diminish in an overload situation which is the case for 128, 256 and 512 tenants. Average throughput halves with each doubling of the number of tenants, but total throughput stays the same meaning that sharing a hardware context between multiple tenants does not incur unacceptable overhead.

The peak performance for a single instance caused by single- threaded serialized transaction execution as employed by current main memory database systems is not a limiting factor here. A single threaded instance for a tenant can execute about 80,000 transactions per second (c.f. Table s1) even when OLAP queries run simultaneously on a snapshot. A back-of-the-envelope calculation shows, that even big retailers like Amazon – on average – have orders of magnitude less orders than can be processed in 80,000 TPC-C transactions: Amazon has a yearly revenue of about 15 billion Euros [KN11]. Assuming that an individual order line is valued at 15 Euros and each order contains an average of 5 items, the average number of orders is less than 7 per second – significantly less than the 80,000 order related transactions achievable in a single thread.

The transaction throughput can be sustained over time and does not degrade for customers with a high amount of data. Figure 3 shows a single instance run at peak performance until the database reached a size of about 900 gigabytes of data. Note that this amount of data can cover orders totaling about 10 billion items which were processed in less than 8 hours.

When the resource requirements of all tenants on a server outgrow the available resources, tenants can be moved easily by leveraging the snapshot mechanism integrated in HyPer, which can be applied to any main memory database system. A consistent snapshot of the DBMS can be created with minimal overhead as described in [KN11], transferred to the target server and be updated using the redo log. That way, migrating a tenant requires only one visible interruption in the order of milliseconds for snapshot creation before clients can start running transactions against the target server.

3 Preliminary results

Our evaluation showing the implementation of multi-tenancy on a many core server uses our HyPer main memory database system. HyPer was originally engineered to show the feasibility of executing both OLTP and OLAP on the same database state by using a virtual memory snapshot (forked process) for the OLAP queries [KN11]. Because HyPer has been built from scratch to specifically work in main memory, its computational as well as memory footprint is small, making it an ideal candidate for the evaluation of a low overhead multi-tenancy approach.

In order to show different multi-tenancy scenarios, we built a management component which can automatically deploy groups of clients with different usage patterns in terms of throughput, memory usage and change in resource consumption over time. We were able to show the viability of different predefined scenarios with changing number of tenants and workload characteristics. In Figure 4, the graphical interface of the management component is shown. The scenario displayed is a server initially hosting 50 tenants with comparable throughput requirements. The fact that one tenants steadily increasing throughput demands have no visible influence on other tenants on the same machine can be observed, proving the high elasticity provided by our setup.

We were able to test the full spectrum of flexibility that a TB server provides: Sharing of the resources by hundreds of small tenants, dedicating the entire server to a large Amazon-style tenant as well as several scenarios in between.

The CH-benCHmark [CFG$^+$11], a combination of the TPC-C transactional workload and the queries specified in the TPC-H, will be used to simulate each tenants workload. The workload can be parameterized to exhibit different characteristics, both in initial throughput and memory consumption as well as their variance over time. This allows for simulating both, OLTP transactions as well OLAP queries executed using HyPer's snapshotting mechanism, therefore simulating the full repertoire of client needs.

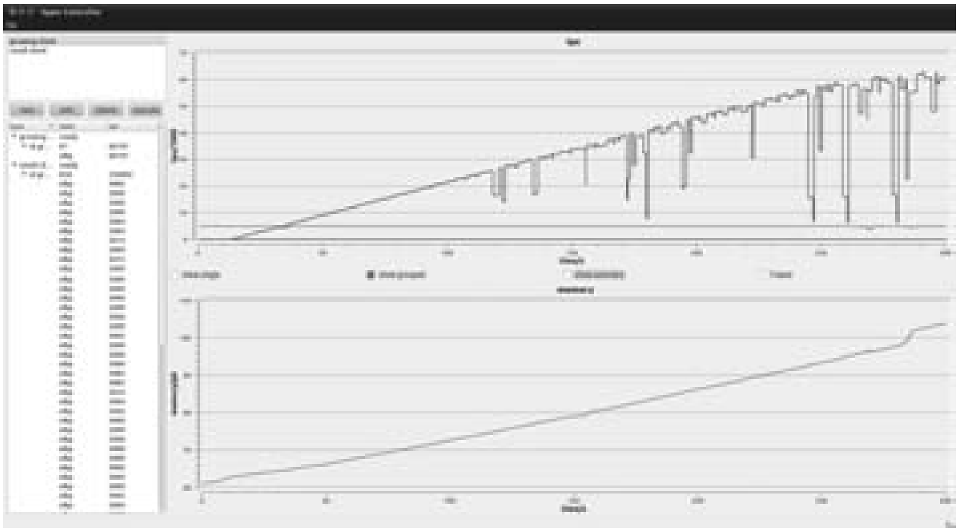

Figure 4: Screenshot displaying the management component of our approach. The diagram in the top-right shows a live view of the transaction throughput of each tenant. Note that the constantly rising transaction rate of the blue tenant does not influence the other 50 instances pictured in red.

4 Summary

Our presentation will show that there are elastic scaling solutions beyond massive deployment of cheap low-end machines. The approach introduced in this paper allows high transaction throughput by collocating many tenants on a multi-core server, allowing cost to be split among tenants and common resources to shared. Our setup behaves predictable even when workloads change drastically and allows a high degree of elasticity for many small tenants with growing data management needs as well as big tenants having to process ten-thousands of transactions.

The advantages of our approach are achieved by combining a multitude of factors:

- Our state of the art, small footprint main memory database system, HyPer. By combining recent research results like serial execution without a a lock- or buffermanager, transaction and query compilation in a database system specifically designed for use in main memory, we constructed a low footprint high performance DBMS that can be individually deployed for each tenant.
- By allowing the operating system to schedule each tenant separately, system resources are used efficiently without explicitly managing all DBMS instances. Furthermore, the architecture automatically benefits from improvements in the operating system, for instance adjustments due to changing hardware properties like - for instance - non uniform memory access.

- Simplified resource sharing like – for example – read-only database content or memory temporarily used for query execution by relinquishing strict separation in contrast to virtualized environments.

All in all, we could show how main memory will not only change the database systems landscape in terms of processing speed but additionally in how solutions for well known problems – like multi-tenancy – can be crafted.

References

[ASJK11] Stefan Aulbach, Michael Seibold, Dean Jacobs, and Alfons Kemper. Extensibility and Data Sharing in evolving multi-tenant databases. In *ICDE*, pages 99–110, 2011.

[CFG+11] Rick Cole, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompaß, Harumi Kuno, Raghunath Nambiar, Thomas Neumann, Meikel Poess, Kai-Uwe Sattler, Michael Seibold, Eric Simon, and Florian Waas. The Mixed Workload CH-benCHmark. In *DBTest*, 2011.

[FKN11] Florian Funke, Alfons Kemper, and Thomas Neumann. HyPer-sonic Combined Transaction AND Query Processing. In *PVLDB*, 2011.

[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):1496–1499, 2008.

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

[Neu11] Thomas Neumann. Efficiently Compiling Efficient Query Plans for Modern Hardware. In *VLDB*, 2011.

[Vol10] VoltDB LLC. VoltDB Technical Overview. `http://voltdb.com/_pdf/VoltDBTechnicalOverviewWhitePaper.pdf`, 2010.