One DBMS for all: the Brawny Few and the Wimpy Crowd

Tobias Mühlbauer
Wolf Rödiger
Robert Seilbeck
Angelika Reiser
Alfons Kemper
Thomas Neumann

Technische Universität München
{muehlbau, roediger, seilbeck, reiser, kemper, neumann}@in.tum.de

ABSTRACT

Shipments of smartphones and tablets with wimpy CPUs are outpacing brawny PC and server shipments by an ever-increasing margin. While high performance database systems have traditionally been optimized for brawny systems, wimpy systems have received only little attention; leading to poor performance and energy inefficiency on such systems.

This demonstration presents HyPer, a high-performance hybrid OLTP&OLAP main memory database system that we optimized for both, brawny and wimpy systems. The efficient compilation of transactions and queries into efficient machine code allows for high performance, independent of the target platform. HyPer has a memory footprint of just a few megabytes, even though it supports the SQL-92 standard, a PL/SQL-like scripting language, and ACID-compliant transactions. It is the goal of this demonstration to showcase the same HyPer codebase running on (a) a wimpy ARM-based smartphone system and (b) a brawny x86-64-based server system. In particular, we run the TPC-C, TPC-H, and a combined CH-benCHmark and report performance and energy numbers. The demonstration further allows the interactive execution of arbitrary SQL queries and the visualization of optimized query plans.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems

Keywords

high performance; energy efficiency; wimpy; brawny

1. INTRODUCTION

Processor shipments reached 1.5 billion units in 2013, a rise of 24% over 2012 [4]. This growth was mainly driven by strong smartphone and tablet sales. PC and server sales, however, stagnated (see Fig. 1). As shipments of wimpy CPUs are outpacing shipments of brawny CPUs, we are entering an era of *the brawny few and the wimpy crowd*.

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

ACM 978-1-4503-2376-5/14/06 ...$15.00.
http://dx.doi.org/10.1145/2588555.2594527.

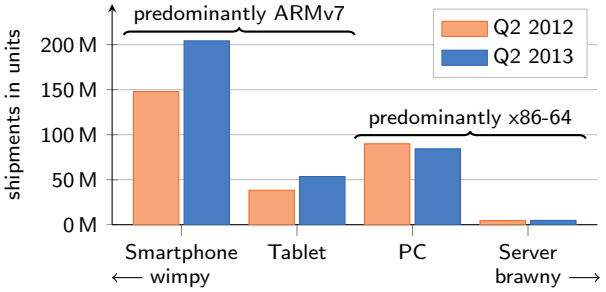
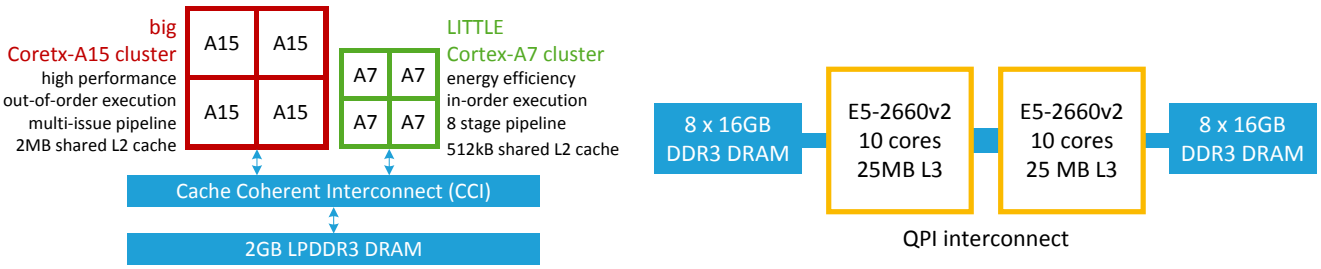


Figure 1: Shipments of wimpy processors are outpacing shipments of brawny processors [4]

While the number of devices with wimpy processors is ever-increasing, these devices receive only little attention from the database community. It is true that database vendors have developed small-footprint database systems such as IBM DB2 Everyplace, Oracle Lite and BerkeleyDB, SAP Sybase SQL Anywhere, and Microsoft SQL Server CE. Yet, these systems either reached end-of-life, are non-relational data stores, or are intended for synchronization with a remote backend server only. In fact, SQLite has evolved to become the de facto standard database for mobile devices. Apple's and Google's mobile operating systems both use it as the default database solution [1, 3]. While this makes SQLite the backbone of most smartphone applications, it neither offers high-performance nor is it specifically optimized for wimpy processors. Our benchmarks (see Sect. 4) show that the performance of SQLite is orders of magnitude slower than an optimized high performance database kernel.

Nonetheless, the need for high-performance database systems on mobile devices is growing. An increasing number of applications run natively on mobile devices and roundtrip latencies to data centers hinder user experience. The development of more disconnected and sophisticated applications thus requires full-featured high-performance data processing capabilities. Besides, energy efficiency is an important factor on mobile devices and usually goes hand in hand with performance [9]. This is because faster data processing consumes less CPU time and modern CPUs can save large amounts of energy using dynamic frequency scaling.

Ideally, a relational database system for both, brawny and wimpy systems, should (i) offer high-performance ACID-compliant transaction and SQL query processing capabilities and (ii) be platform independent such that the system is universally deployable and only one codebase needs to be maintained. Further, mobile and embedded devices require a database system with a small memory footprint.



(a) wimpy: ARMv7 big.LITTLE system

(b) brawny: x86-64 NUMA system

Figure 2: Demonstration platforms: (a) a wimpy smartphone system and (b) a brawny server system

HyPer [5], our high-performance hybrid OLTP&OLAP main memory database system, aims at fulfilling these requirements. This demonstration presents our lean HyPer system and showcases its high performance on both, a wimpy smartphone system and a brawny server system (see Fig. 2).

2. THE HYPER MAIN MEMORY DBMS

HyPer [5] is a high-performance relational main memory database system that belongs to an emerging class of hybrid databases, which enable real-time business intelligence by evaluating OLAP queries directly in the transactional database. In HyPer, long-running OLAP is decoupled from mission-critical OLTP using an efficient operating system and hardware-supported snapshotting mechanism based on the POSIX system call `fork`. As other high-performance main memory database systems, HyPer eliminates the ballast caused by buffer management, locking, and latching. This enables serial transaction processing at high speed. With its advanced query optimizer, HyPer further achieves superior query response times comparable (often even superior) to those of MonetDB and Vectorwise, two state-of-the-art analytical main memory databases. Even though the SQL-92 standard, a PL/SQL-like scripting language, and ACID-compliant transaction processing are supported, HyPer has a memory footprint of just a few megabytes.

3. DATA-CENTRIC CODE GENERATION

Most database systems translate incoming queries and transactions into a physical algebra expression and evaluate this expression using the iterator model. Every physical algebraic operator produces a tuple stream from its input and exposes this stream via an iterator, i.e., a function that fetches the next tuple. Despite being convenient and feeling natural, the iterator model is also very slow on modern pipelined CPUs due to a great many (virtual) function calls, degraded branch prediction, and poor code locality. These negative properties of the iterator model are reinforced by the advent of main memory database systems like HyPer, where query and transaction performance is more and more determined by raw CPU costs rather than I/O speed.

To deal with the issues described for the iterator model, several modern database systems such as MonetDB and Vectorwise produce more than one tuple during an iterator call or even all tuples at once. While this kind of block-oriented processing reduces the overhead of function calls and allows for the efficient use of vectorization instructions, it also eliminates the possibility to pipeline data, i.e., passing data from one operator to its parent without copying or materialization; severely limiting peak performance.

HyPer uses a different query evaluation strategy [7] to deal with the shortcomings of the iterator model. We came to the conclusion that it is not necessarily a good idea to exhibit the algebraic operator structure during query processing itself. In HyPer, query processing is thus data-centric rather than operator-centric. Operator boundaries are blurred to enable pipelining and keep data in CPU registers as long as possible. To improve code and data locality, data is further pushed towards consuming operators rather than being pulled. Finally, to achieve optimal performance and get most of the mileage out of a given processor, queries are compiled to optimized native machine code instead of using an interpreter.

More specifically, query compilation in HyPer is based on the LLVM compiler framework and proceeds in three steps: First, incoming queries and transactions are parsed and an algebraic expression is generated and optimized. Second, platform-independent LLVM assembly code is generated based on the optimized algebraic expression. The code generator mimics a producer/consumer interface, where data is taken out of a pipeline breaker and is materialized into the next pipeline breaker. Complex operators, e.g., index logic, are pre-compiled and calls to these operators are generated dynamically during code generation. Third, the generated LLVM assembly code is executed using the optimizing LLVM JIT compiler, which quickly produces extremely fast machine code; usually within a few milliseconds. The LLVM compiler makes our query compilation approach portable, as platform-dependent machine code is generated only in the final step. LLVM backends exist for several instruction sets, e.g., x86, x86-64, and ARM. In case an instruction set is not supported, the LLVM interpreter can be used as a fallback.

Beginning with version 3.4 (December 2013), LLVM offers more reliable and robust machine code compilation for non-x86 platforms using the MCJIT compiler, which is also used by Clang. For this demonstration, we thus ported our query compilation from the legacy JIT compiler to MCJIT in order to allow for the same codebase of HyPer to run on a brawny x86-64 server system and a wimpy ARMv7 smartphone system. On both systems, sources were compiled using LLVM/Clang version 3.4.

4. DEMONSTRATION

This demonstration of the HyPer system exposes the key features of our platform-independent query compilation for wimpy and brawny systems and provides ways of visualizing performance and energy numbers for transactional, analytical, and combined workloads. Size and composition of the workloads are varied during the demonstration. To better demonstrate our data-centric code generation, generated

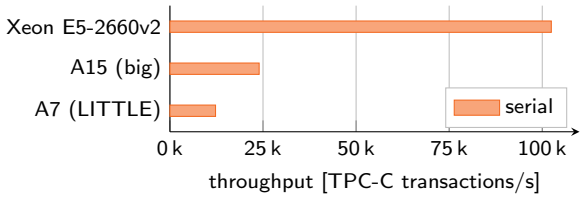


Figure 3: TPC-C throughput of the demo systems

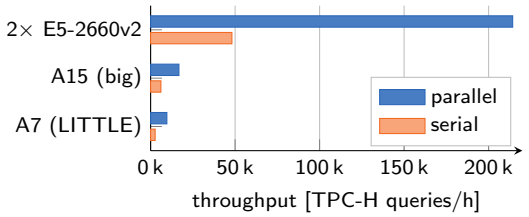


Figure 4: TPC-H throughput of the demo systems

LLVM and native assembly codes are shown and explained. To showcase the full capabilities of HyPer, we provide an interactive SQL editor in which arbitrary SQL-92 queries can be executed and optimized query plans can be visualized. The user interface for this demonstration is implemented as an extension of the HyPer WebInterface[1].

All benchmarks and interactive queries are executed simultaneously on two systems, (a) a wimpy ARMv7 system and (b) a brawny x86-64 system (see Fig. 2):

Wimpy ARMv7 system. The wimpy system is an ARM development board. The board's hardware resembles the one in the Samsung Galaxy S4, a state-of-the-art smartphone. It features a Samsung Exynos5 Octa 5410 CPU, which is based on the ARM big.LITTLE architecture and combines an energy-efficient quad-core ARM Cortex-A7 cluster with a high-performance quad-core ARM Cortex-A15 cluster. Both clusters have highly different characteristics (see Fig. 2(a)). The clusters and a 2 GB LPDDR3 DRAM module are connected via a cache coherent interconnect. A 1 TB external hard disk is attached to the development board via USB 3.0. The system runs a customized Linux kernel version 3.4. During the demonstration, the development board is located at the demo site. To enable energy measurements, the development board is connected to a power supply that collects energy numbers. The power monitor has a sampling rate of 10 Hz and a tolerance of 2%. It exposes its collected data via USB to the development board from where it is pushed towards the demonstration user interface.

Brawny x86-64 system. The brawny server system is a 2-socket Intel Xeon E5-2660v2 non-uniform memory access (NUMA) system with 20 cores and 256 GB of main memory[2]. During the demonstration, the server is located at TUM in Munich, Germany. The system runs a Linux kernel version 3.11. For energy metrics, we use the running average power limit (RAPL) energy counters of the Intel CPUs. With these counters we can record the power consumption of the CPUs and of main memory, which make up a great fraction of the overall energy consumption of the system.

The demonstration consists of three parts. First, a poster is used to raise awareness for the growing number of devices with wimpy processors. Performance of current database systems running on such devices is discussed and opportunities for performance and energy efficiency improvements are presented. Additionally, the HyPer system, its query compilation, and the demonstration systems are introduced.

[1] `http://www.hyper-db.com/interface.html`

[2] Each E5-2660v2 CPU has 10 cores, 20 hardware threads, 25 MB of last level L3 cache and 128 GB DDR3 DRAM.

In the second part, three benchmarks are run to showcase the performance and energy efficiency of the demonstration systems: the TPC-C, the TPC-H, and the combined CH-benCHmark [2]. All reported performance and power measurements are an average over multiple runs.

TPC-C. We run the on-line transaction processing benchmark TPC-C with 5 warehouses and no wait times. Fig. 3 shows the serial execution throughput of the demo systems. As expected, the brawny E5-2660v2 server CPU has a much higher single core peak performance than the wimpy CPUs. Yet, close to 25 k TPC-C transactions per second can be executed on the wimpy system. Regarding performance per Watt, the LITTLE A7 CPU processes 2.8 k transactions per second per Watt and the big A15 CPU processes 10.4 k transactions per second per Watt. SQLite is orders of magnitude slower and less energy efficient.

TPC-H. We run the 22 TPC-H queries on a scale factor 1 database. Fig. 4 reports the throughput numbers for single-threaded and intra-query parallel execution. Regarding performance per Watt, the LITTLE A7 CPU processes 1.2 k (3.7 k parallel) queries per hour per Watt and the big A15 CPU processes 1.5 k (2.2 k parallel) queries per hour per Watt. SQLite is again orders of magnitude slower: SQLite needed 5.6 hours to complete a single TPC-H run and used 131 kJ corresponding to a mere 0.6 queries per hour per Watt. While the scale factor 1 data set was the largest to fit in the 2 GB of main memory on the wimpy system, the brawny system could obviously handle much larger data sets. Regarding performance per Watt, the brawny system processes 0.5 k queries per hour per Watt (1.7 k parallel).

CH-benCHmark. HyPer is a hybrid database system and offers the possibility of evaluating OLAP queries on recent snapshots of the transactional database. The CH-benCHmark benchmarks the transactional and analytical performance of such a hybrid database system. In the CH-benCHmark a transactional workload based on the order entry processing of TPC-C and a corresponding TPC-H-equivalent OLAP query suite run in parallel on the same tables in a single database system.

In the final part of the demonstration, users are welcome to interactively try out the HyPer database by themselves. Fig. 5 shows our SQL editor in which arbitrary queries on the TPC-H and CH-benCHmark schemas can be composed. Queries are executed on both, the brawny and the wimpy system. The user interface reports response times and energy numbers. Besides query execution, optimized query

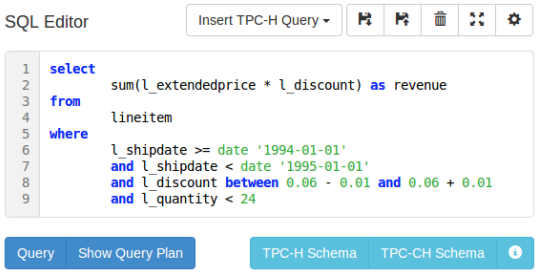


Figure 5: Interactive query editor (Q 6 of TPC-H)

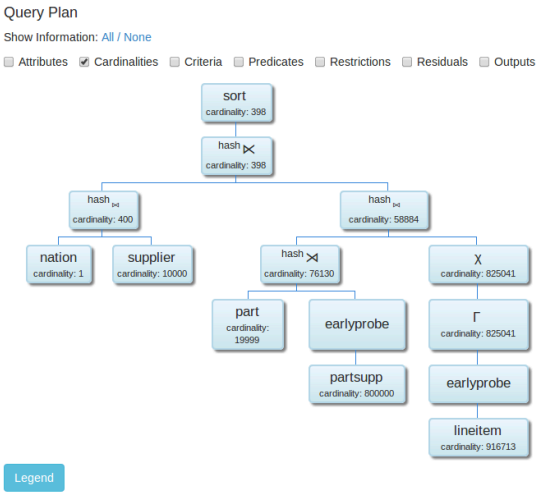


Figure 6: Query plan visualization (Q 20 of TPC-H)

plans can be visualized (see Fig. 6). The visualization shows, at a very detailed level, the operator structure, cardinality estimates, join types, and other information of the plan. Users can further repeat the benchmarks by choosing a workload (TPC-C, TPC-H, and CH-benCHmark) and configuring its parameters such as the number of concurrent query streams (see top of Fig. 7). Throughout the demonstration, energy numbers for the wimpy system are displayed in an energy monitor (see bottom of Fig. 7).

5. TAKE-AWAY MESSAGE

Shipments of wimpy devices such as smartphones and tablets are outpacing shipments of brawny PC and server systems. With this demonstration we intend to raise the awareness of the database community to focus not only on optimizing performance on brawny, but also on wimpy systems. In particular, we demonstrate HyPer, a hybrid OLTP &OLAP main memory database system with a small memory footprint. We highlight that its data-centric code generation and platform-optimized machine code compilation allow for a lean database system that achieves high performance on both, brawny and wimpy systems; even for different CPU architectures. Thereby we get most of the mileage out of a given processor. We further show that high performance directly translates to high energy efficiency, which is particularly important on energy-constrained devices such as smartphones and tablets. The platform-agnostic code generation allows the HyPer system to maintain a single codebase for multiple platforms. HyPer's performance and versatile usability enable richer data processing capabilities for more sophisticated mobile applications. Additionally, one platform-independent database system optimized for brawny and wimpy systems enables distributed database systems like our scaled-out version of HyPer [6] and energy-optimized systems like WattDB [8] to be composed of heterogeneous nodes, each carefully selected for different workloads and quality of service requirements.

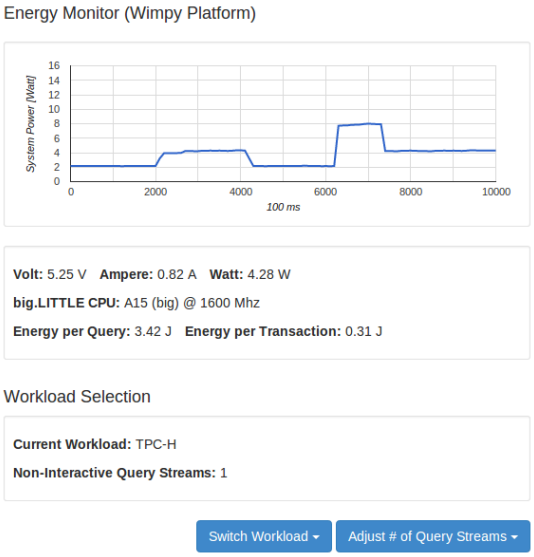


Figure 7: Energy monitor and workload selection

6. ACKNOWLEDGEMENTS

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.

7. REFERENCES

[1] Apple. Data Management in iOS. `https://developer.apple.com/technologies/ios/data-management.html`.

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

[3] Google. Storage Options. `http://developer.android.com/guide/topics/data/data-storage.html#db`.

[4] IHS. Processor Market Set for Strong Growth in 2013, Courtesy of Smartphones and Tablets. `http://press.ihs.com/printpdf/18632`.

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

[6] T. Mühlbauer, W. Rödiger, A. Reiser, A. Kemper, and T. Neumann. ScyPer: Elastic OLAP throughput on transactional data. In *DanaC*, 2013.

[7] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9), 2011.

[8] D. Schall and T. Härder. Energy-proportional query execution using a cluster of wimpy nodes. In *DaMoN*, 2013.

[9] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *SIGMOD*, 2010.