HyPer-sonic Combined Transaction AND Query Processing

Thomas Neumann

Technische Universität München

October 26, 2011

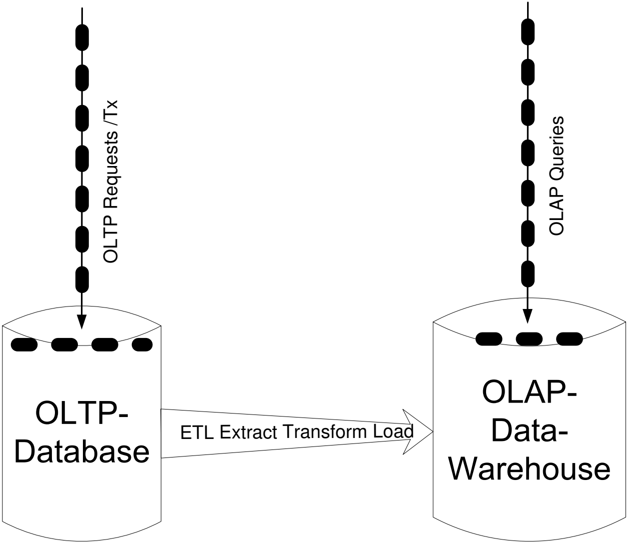
Motivation - OLTP vs. OLAP

OLTP and OLAP have very different requirements

- OLTP
  - high rate of small/tiny transactions
  - high locality in data access
  - update performance is critical
- OLAP
  - few, but long running transactions
  - aggregates large parts of the database
  - must see a consistent database state the whole time

Traditionally, DBMSs either good at OLTP or good at OLAP

Motivation - Traditional Solution



not very satisfying. stale data, redundancy, etc.

Motivation - Hardware Trends

Intel
Tera Scale Initiative
Server with 1 TB main memory
ca. 40K Euro from Dell

- main memory grows faster than (business) data
- can afford to keep data in memory
- memory is not just a fast disk
- should make use of this facts

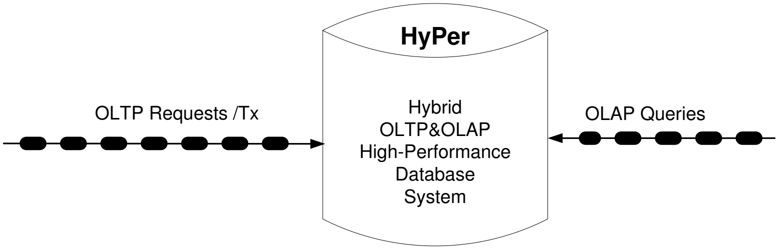
Amazon

Data Volume
Revenue: 25 billion Euro
Avg. Item Price: 15 Euro
ca. 1.6 billion order lines per year
ca. 54 Bytes per order line
ca. 90 GB per year
+ additional data - compression

Transaction Rate
Avg: 32 orders per s
Peak rate: Thousands/s
+ inquiries

HyPer

Our system



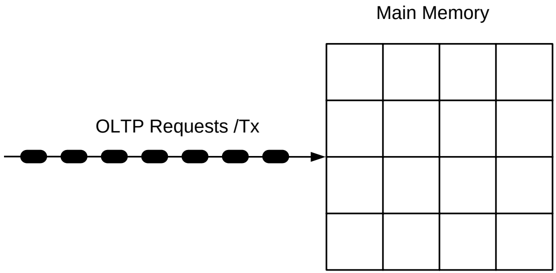
Combined OLTP/OLAP system using modern hardware

HyPer - Design

- OLTP performance is crucial
- avoid anything that would slow down OLTP
- OLTP should operate as if there were no OLAP
- OLAP is not that performance sensitive, but needs consistency
- locking/latching is out of question (OLAP would slow down OLTP)

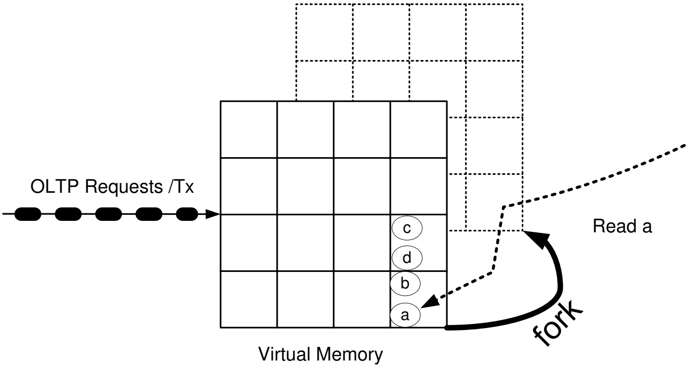
Idea: we are a main memory database. Use hardware support.

HyPer - Pure OLTP workload



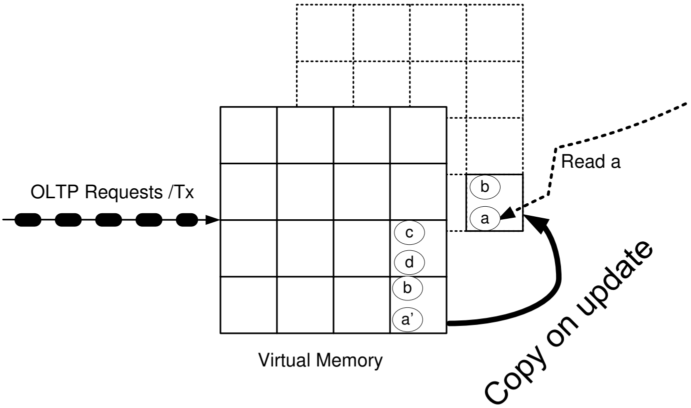
- purely main memory, OLTP transactions need a few μs
- can afford serial execution of transactions (at least initially)
- avoids any concurrency issues

HyPer - Virtual Memory Supported Snapshots



- OLAP sessions need a consistent snapshot over a relatively long time
- use the MMU / OS support to separate OLTP and OLAP
- the *fork* separates OLTP from OLAP, even though they are initially the same

HyPer - Copy on Update



- the MMU detects writes to shared data
- modified pages are copied, both parts have unique copies afterwards
- avoids any interaction between OLTP and OLAP
- like an ultra-efficient shadow paging without the disadvantages

HyPer - Snapshots

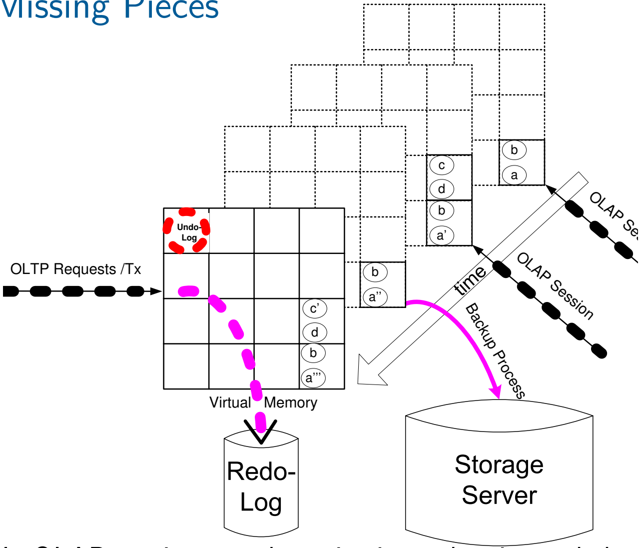
We use *fork* to create transaction consistent snapshots

- each OLAP sessions sees one certain point in time
- can do long-running aggregates/analysis
- the data (apparently) stays the same
- if it changes, the MMU makes sure that OLAP does not notice
- eliminates need for latching/locking

And *fork* is cheap!

- only the page table is copied, not the pages themselves
- some care is needed to scale to large memory sizes
- but can *fork* 40GB in 2.7ms

HyPer - Missing Pieces



- multiple OLAP sessions, each copies just what is needed
- logging is needed for ACID properties
- backups for fast restart

Data-Centric Query Execution

HyPer does not use the classical iterator model

Why does the iterator model (and its variants) use the operator structure for execution?

- it is convenient, and feels natural
- the operator structure is there anyway
- but otherwise the operators only describe the data flow
- in particular operator boundaries are somewhat arbitrary

What we really want is **data centric** query execution

- data should be read/written as rarely as possible
- data should be kept in CPU registers as much as possible
- the code should center around the data,
  not the data move according to the code
- increase locality, reduce branching

Data-Centric Query Execution (2)

Processing is oriented along pipeline fragments.

Corresponding code fragments:

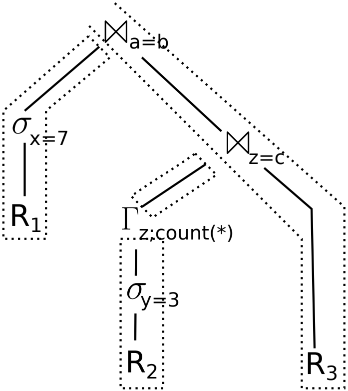
initialize memory of $\bowtie_{a=b}$, $\bowtie_{c=z}$, and Γ_z

for each tuple t in R_1
    if $t.x = 7$
        materialize t in hash table of $\bowtie_{a=b}$

for each tuple t in R_2
    if $t.y = 3$
        aggregate t in hash table of Γ_z

for each tuple t in Γ_z
    materialize t in hash table of $\bowtie_{z=c}$

for each tuple t_3 in R_3
    for each match t_2 in $\bowtie_{z=c}[t_3.c]$
        for each match t_1 in $\bowtie_{a=b}[t_3.b]$
            output $t_1 \circ t_2 \circ t_3$



Data-Centric Query Execution (3)

The algebraic expression is translated into query fragments.

Each operator has two interfaces:

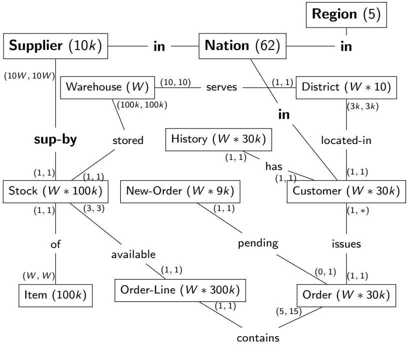
1. produce
   - asks the operator to produce tuples and push it into
2. consume
   - which accepts the tuple and pushes it further up

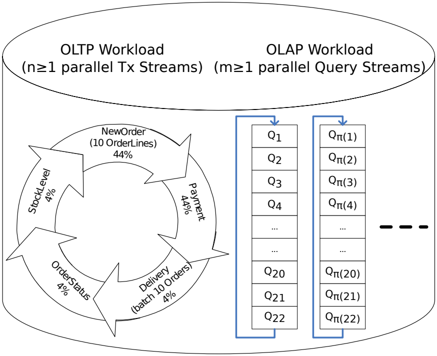
Note: only a mental model!

- the functions are not really called
- they only exist conceptually during code generation
- each “call” generates the corresponding code
- operator boundaries are blurred, code centers around data
- we generate machine code at compile time
- initially using C++, now using LLVM

Evaluation

We used a combined TPC-C and TPC-H benchmark (12 warehouses)





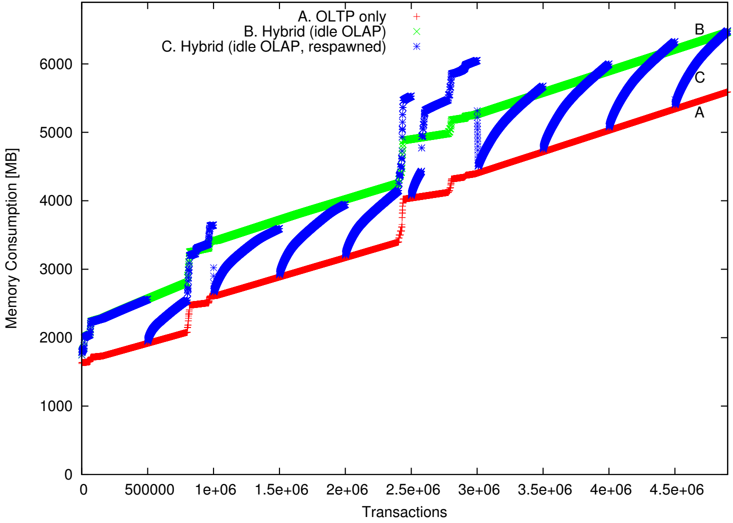
- TPC-C transactions are unmodified
- TPC-H queries adapted to the combined schema
- OLTP and OLAP runs in parallel

TPC-C+H Performance

	HyPer configurations				MonetDB	VoltDB
	one query session (stream) single threaded OLTP		3 query sessions (streams) 5 OLTP threads		no OLTP 1 query stream	no OLAP only OLTP
Query No.	OLTP throughput	Query resp. times (ms)	OLTP throughput	Query resp. times (ms)	Query resp. times (ms)	results from VoltDB web page
Q1	new order: 56961 tps; total: **126576** tps	67	new order: 171384 tps; total: **380868** tps	71	63	**55000** tps on single node; **300000** tps on 6 nodes
Q2		163		212	210	
Q3		66		73	75	
Q4		194		226	6003	
Q5		1276		1564	5930	
Q6		9		17	123	
Q7		1151		1466	1713	
Q8		399		593	172	
Q9		206		249	208	
Q10		1871		2260	6209	
Q11		33		35	35	
Q12		156		170	192	
Q13		185		229	284	
Q14		122		156	722	
Q15		528		792	533	
Q16		1353		1500	3562	
Q17		159		168	342	
Q18		108		119	2505	
Q19		103		183	1698	
Q20		114		197	750	
Q21		46		50	329	
Q22		7		9	141	

Dual Intel X5570 Quad-Core-CPU, 64GB RAM, RHEL 5.4

Memory Consumption



- we only have to replicate the working set

Conclusion

- main memory databases change the game
- very high throughput, transactions should never wait
- minimize latching and locks to get best performance
- use MMU support instead to separate OLTP and OLAP
- compiled, data-centric queries for excellent performance

HyPer is a very fast hybrid OLTP/OLAP system

- top performance for both OLTP and OLAP
- full ACID support

It is indeed possible to build a combined OLTP/OLAP system!