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.

ETL Extract Transform Load

OLTP Requests /Tx

OLAP Queries

OLTP-Database

OLAP-Warehouse

ETL Extract Transform Load
Motivation - Hardware Trends

Intel
Tera Scale Initiative
Server with 1 TB main memory
c.a. 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
c.a. 1.6 billion order lines per year
c.a. 54 Bytes per order line
c.a. 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 $\mu$s
- 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
• 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
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
• 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 \( \Join_{a=b} \), \( \Join_{c=z} \), and \( \Gamma_z \)

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

for each tuple \( t \) in \( R_2 \)
  if \( t.y = 3 \)
    aggregate \( t \) in hash table of \( \Gamma_z \)

for each tuple \( t \) in \( \Gamma_z \)
  materialize \( t \) in hash table of \( \Join_{z=c} \)

for each tuple \( t_3 \) in \( R_3 \)
  for each match \( t_2 \) in \( \Join_{z=c} [t_3.c] \)
    for each match \( t_1 \) in \( \Join_{a=b} [t_3.b] \)
      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

<table>
<thead>
<tr>
<th>Query No.</th>
<th><strong>HyPer</strong> configurations</th>
<th><strong>MonetDB</strong></th>
<th><strong>VoltDB</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>one query session (stream)</td>
<td>3 query sessions (streams)</td>
<td>no OLTP no OLTP</td>
</tr>
<tr>
<td></td>
<td>single threaded OLTP OLTP throughput</td>
<td>5 OLTP threads OLTP throughput</td>
<td>1 query stream Query resp. times (ms)</td>
</tr>
<tr>
<td>Query resp. times (ms)</td>
<td>Query resp. times (ms)</td>
<td>63</td>
<td>75</td>
</tr>
<tr>
<td>Q1</td>
<td>67</td>
<td>71</td>
<td>63</td>
</tr>
<tr>
<td>Q2</td>
<td>163</td>
<td>212</td>
<td>210</td>
</tr>
<tr>
<td>Q3</td>
<td>66</td>
<td>73</td>
<td>75</td>
</tr>
<tr>
<td>Q4</td>
<td>194</td>
<td>226</td>
<td>6003</td>
</tr>
<tr>
<td>Q5</td>
<td>1276</td>
<td>1564</td>
<td>5930</td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td>17</td>
<td>123</td>
</tr>
<tr>
<td>Q7</td>
<td>1151</td>
<td>1466</td>
<td>1713</td>
</tr>
<tr>
<td>Q8</td>
<td>399</td>
<td>593</td>
<td>172</td>
</tr>
<tr>
<td>Q9</td>
<td>206</td>
<td>249</td>
<td>208</td>
</tr>
<tr>
<td>Q10</td>
<td>1871</td>
<td>2260</td>
<td>6209</td>
</tr>
<tr>
<td>Q11</td>
<td>33</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>Q12</td>
<td>156</td>
<td>170</td>
<td>192</td>
</tr>
<tr>
<td>Q13</td>
<td>185</td>
<td>229</td>
<td>284</td>
</tr>
<tr>
<td>Q14</td>
<td>122</td>
<td>156</td>
<td>722</td>
</tr>
<tr>
<td>Q15</td>
<td>528</td>
<td>792</td>
<td>533</td>
</tr>
<tr>
<td>Q16</td>
<td>1353</td>
<td>1500</td>
<td>3562</td>
</tr>
<tr>
<td>Q17</td>
<td>159</td>
<td>168</td>
<td>342</td>
</tr>
<tr>
<td>Q18</td>
<td>108</td>
<td>119</td>
<td>2505</td>
</tr>
<tr>
<td>Q19</td>
<td>103</td>
<td>183</td>
<td>1698</td>
</tr>
<tr>
<td>Q20</td>
<td>114</td>
<td>197</td>
<td>750</td>
</tr>
<tr>
<td>Q21</td>
<td>46</td>
<td>50</td>
<td>329</td>
</tr>
<tr>
<td>Q22</td>
<td>7</td>
<td>9</td>
<td>141</td>
</tr>
</tbody>
</table>

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

Thomas Neumann

HyPer

50000 tps on 6 nodes
• 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!