Rack-scale Data Processing System

Jana Giceva, Darko Makreshanski, Claude Barthels, Alessandro Dovis, Gustavo Alonso

Systems Group, Department of Computer Science, ETH Zurich
Rack-scale Data Processing System

Jana Giceva, Darko Makreshanski, Claude Barthels, Alessandro Dovis, Gustavo Alonso

Systems Group, Department of Computer Science, ETH Zurich
FruitBox – a data processing system

Transactionals QP

Graph processing

Analytical QP

Machine learning

Ad-hoc BI QP

MULTI FLAVOR DATA PROCESSING
FruitBox

Building a system for multi-flavor data processing:

1. Hardware that meets the resource demand.

2. System architecture to support workload heterogeneity.

3. Aim for 10s-100s millions of requests per second.

4. Efficient resource utilization.
FruitBox – a **rack-scale** data processing system

- Which box could run such a heterogeneous WL?
  - A multicore is not enough

- A rack-scale system:
  - More resources
  - Better isolation
  - Blurring the machine-cluster boundaries
Rack-scale data processing system

Custom build a rack-scale system for data processing?

Many such commercial systems exist – Data Appliances

ORACLE® Exadata  
Netezza (IBM) TwinFin  
SAP HANA  
and many more ...
System design for Multi-flavor data processing

- Separate data-storage from data-processing

- Achieve both *physical* and *logical* data independence
Storage Engine

Data processing
- Transactional processing
- Analytical processing
- Machine Learning

Tuple- and batch-based interface to the storage engine.
Storage Engine

Data processing

Transactional processing
Analytical processing
Machine Learning

Storage Engine components:
- KV Stores (B-tree)
- Crescando Scans

KV Stores → transactional
Scans → analytical

Tuple- and batch-based interface to the storage engine.
Storage Engine

Data processing
- Transactional processing
- Analytical processing
- Machine Learning

Storage engine components:
- KV Stores (B-tree)
- Crescando Scans

KV Stores → transactional
Scans → analytical

Transaction logic separated from query processing.

Tuple- and batch-based interface to the storage engine.

Hyder [SIGMOD’15], HyPer[VLDB’11], Hekaton [SIGMOD’14], SharedDB [Giannikis PhD’14], Tell [SIGMOD’15], Multimed [Eurosys’11]
Handling millions of requests/second

- It makes no sense to process them individually if they access the same data.
- Why should each query scan a TB of data?

- **Batch requests** – share data, computation, bandwidth ... for higher throughput and predictable performance trading off a bit of latency.

IBM Blink, MonetDB/X100[VLDB’07], CJOIN [VLDB’09], Crescando[VLDB’09], SharedDB [VLDB’12,’14],

Workshop for Rack-scale Computing
Efficient resource utilization

MULTI FLAVOR DATA PROCESSING

- Graph processing
- Machine learning
- Transactional QP
- Analytical QP
- Ad-hoc BI QP

Getting the most out of such a complex system requires cross-layer optimization.
- e.g. DB/OS co-design

Already some work on multicore systems.

- Noisy system environment
- Load interaction

- Unpredictable performance
- Not meeting SLAs

- Resource overprovisioning
- Inefficiency and higher cost
COD: DB/OS co-design

What is the knowledge we have?
Who knows what?

Big semantic gap!

Application requirements and characteristics

Hardware & architecture + System state and utilization of resources

COD: Database/Operating System co-design [CIDR’12]

Workshop for Rack-scale Computing
COD’s interface

DBMS

DB storage engine

Notification on updates

Explicit allocation

DB/OS Interface

Constraints and requirements

OS policy engine

OS

other apps

Notification on updates

Explicit allocation
Adaptability to dynamic system state

Experiment setup
- AMD MagnyCours
- 4 x 2.2GHz AMD Opteron 6174 processors
- total Datastore size 53GB
- Noise: another CPU-intensive task running on core 0

### Adaptability – Latency

- **Naïve datastore engine**
- **SLA**
- **COD**
Resource efficient deployment

Deployment of operators to CPU cores

Deployment algorithm

Resource requirements of operators

Multicore machine

Resource Activity Vectors

Model of multicore machine

Data dependency graph

Query plan

DB

OS

Deployment of query plans on multicores [VLDB’15]

Workshop for Rack-scale Computing
Evaluation

Query plan

- SharedDB’s TPC-W [1]
- 11 web-interactions in one query plan
- 44 operators
- 20GB dataset

AMD Magnycours

- 4 x 2 dies:
  - 6 cores
  - 5 MB L3 cache
  - 16 GB NUMA node

[1] SharedDB – Giannikis et al. VLDB’12
## Comparison with standard approaches

<table>
<thead>
<tr>
<th>Approaches</th>
<th># cores</th>
<th>Throughput [WIPS]</th>
<th>Response Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Stdev</td>
</tr>
<tr>
<td>Default OS</td>
<td>48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operator per core</td>
<td>44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deployment algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison with standard approaches

<table>
<thead>
<tr>
<th>Approaches</th>
<th># cores</th>
<th>Throughput [WIPS]</th>
<th>Response Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Stdev</td>
</tr>
<tr>
<td>Default OS</td>
<td>48</td>
<td>317.30</td>
<td>31.11</td>
</tr>
<tr>
<td>Operator per core</td>
<td>44</td>
<td>425.86</td>
<td>54.34</td>
</tr>
<tr>
<td>Deployment algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Comparison with standard approaches

<table>
<thead>
<tr>
<th>Approaches</th>
<th># cores</th>
<th>Throughput [WIPS]</th>
<th>Response Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Stdev</td>
</tr>
<tr>
<td>Default OS</td>
<td>48</td>
<td>317.30</td>
<td>31.11</td>
</tr>
<tr>
<td>Operator per core</td>
<td>44</td>
<td>425.86</td>
<td>54.34</td>
</tr>
<tr>
<td>Deployment algorithm</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Workshop for Rack-scale Computing
**Comparison with standard approaches**

<table>
<thead>
<tr>
<th>Approaches</th>
<th># cores</th>
<th>Throughput [WIPS]</th>
<th>Response Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Stdev</td>
</tr>
<tr>
<td>Default OS</td>
<td>48</td>
<td>317.30</td>
<td>31.11</td>
</tr>
<tr>
<td>Operator per core</td>
<td>44</td>
<td>425.86</td>
<td>54.34</td>
</tr>
<tr>
<td>Deployment algorithm</td>
<td>6</td>
<td>428.07</td>
<td>32.80</td>
</tr>
</tbody>
</table>
Comparison with standard approaches

<table>
<thead>
<tr>
<th>Approaches</th>
<th># cores</th>
<th>Throughput [WIPS]</th>
<th>Response Time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>Stdev</td>
</tr>
<tr>
<td>Default OS</td>
<td>48</td>
<td>317.30</td>
<td>31.11</td>
</tr>
<tr>
<td>Operator per core</td>
<td>44</td>
<td>425.86</td>
<td>54.34</td>
</tr>
<tr>
<td>Deployment algorithm</td>
<td>6</td>
<td>428.07</td>
<td>32.80</td>
</tr>
</tbody>
</table>

Performance / Resource efficiency savings of $x7.37$
Conclusion

Multi-flavor data processing system

- We have all the pieces of the puzzle

Separate data-storage from data-processing

Efficient resource management

Batching as a first class citizen

... on a rack-scale system

Putting them together opens a lot of opportunities.
Conclusion

Multi-flavor data processing system

- We have all the pieces of the puzzle

- Intelligent storage engine:
  - Co-processors, active-memory, hardware specialization (FPGAs)

- Optimizing the network stack:
  - ... for different memory access patterns

- Extend the cross-layer interface:
  - DB optimizer that is aware of the complexity of the rack

- Rack-scale resource management

Putting them together opens a lot of opportunities.