Rackscale - the things that matter

GUSTAVO ALONSO
SYSTEMS GROUP
DEPT. OF COMPUTER SCIENCE
ETH ZURICH
On the way to VU, this morning ...

One size does not fit all?
• Intelligent storage manager
• Massive caching
• RAC based architecture
• Fast network interconnect
NETEZZA (IBM) TWINFIN

• No storage manager
• Distributed disks (per node)
• FPGA processing
• No indexing
Hardware rules

- Multicore, Many core
- Transactional Memory
- SIMD, AVX, vectorization
- SSDs, persistent memory
- Infiniband, RDMA
- GPUs, FPGAs (hardware acceleration)
- Intelligent storage engines, main memory
- Database appliances

- Reacting to changes we do not control
What does it mean?

- Homogeneous inside
  - The components will still be mostly general purpose
  - Economies of scale
- Heterogeneous outside
  - Systems tailored to the application
  - Performance through customization
Multicore is great: avoid distribution

Nobody ever got fired for using Hadoop on a Cluster
HotCDP 2012, Bern, Switzerland

- Analysis of MapReduce workloads:
  - Microsoft: median job size < 14 GB
  - Yahoo: median job size < 12.5 GB
  - Facebook: 90% of jobs less than 100 GB
- Fit in main memory
- One server more efficient than a cluster
- Adding memory to a big server better than using a cluster
Where is the heterogeneity?
The take away message

- Easy to build a customized system addressing one use case
  - Less and less interesting
- Difficult to design techniques and tools for developing customized systems
  - Increasingly relevant
What matters

- Hierarchical, heterogeneous processors
  - Processing at all levels
- Using the hardware, knowing the load
  - Determining what to run where
- The case for sharing
  - Batch processing rather than single jobs
- It is the data, stupid
  - What a system can do and what it cannot do
Hierarchical, heterogeneous systems
In the future ... 

- Expect hardware acceleration everywhere:
  - Co-processors
  - Intelligent storage
  - Intelligent (active) memory
  - In-network data processing
  - Hierarchical configurations to manage complexity
SELECT customer_name
FROM cells
WHERE amount > 200;

1. Smart Scan Constructed And Sent To Cells
2. Consolidated Result Set Built From All Cells
3. Rows Returned
4. Only the relevant columns
   - customer_name
   - required rows
   - where amount>200
   - are returned to hosts

- CPU consumed by predicate evaluation is offloaded
- Moving scan processing off the database host frees host CPU cycles and eliminates massive amounts of unproductive messaging
  - Returns the needle, not the entire hay stack
Hardware might solve your problem

Louis Woods, Gustavo Alonso, Jens Teubner: Parallel Computation of Skyline Queries. FCCM 2013
Ibex = Intelligent storage engine
Inserting the FPGA in the data path

- SSD → raw data → FPGA → filtered data → host system (MySQL)
Engine design
So far so good
Points of interest

<table>
<thead>
<tr>
<th>Query/Storage Engine</th>
<th>( \Delta )-Power</th>
<th>Energy Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point Query / MyISAM</td>
<td>22 watts</td>
<td>864 joules</td>
</tr>
<tr>
<td>Point Query / INNODB</td>
<td>24 watts</td>
<td>7380 joules</td>
</tr>
<tr>
<td>Point Query / Ibex</td>
<td>3 watts</td>
<td>216 joules</td>
</tr>
<tr>
<td>Hybrid Join / MyISAM</td>
<td>22 watts</td>
<td>864 joules</td>
</tr>
<tr>
<td>Hybrid Join / INNODB</td>
<td>24 watts</td>
<td>7380 joules</td>
</tr>
<tr>
<td>Hybrid Join / Ibex</td>
<td>3 watts</td>
<td>216 joules</td>
</tr>
<tr>
<td>Group By / MyISAM</td>
<td>22 watts</td>
<td>864 joules</td>
</tr>
<tr>
<td>Group By / INNODB</td>
<td>24 watts</td>
<td>7380 joules</td>
</tr>
<tr>
<td>Group By / Ibex</td>
<td>3 watts</td>
<td>216 joules</td>
</tr>
</tbody>
</table>

**CPU usage when executing GROUP BY**

**INNODB**

- CPU Usage
- CPU Usage History

**Ibex**

- CPU Usage
- CPU Usage History
Characterizing hardware and loads
Deployment and scheduling

- The times of over provisioning are over:
  - Too expensive
  - No longer politically correct
  - No switch on and off (too expensive)

- Dynamic deployment and scheduling
  - More complex loads
  - More data movement
  - More heterogeneous hardware
Heterogeneity is a mess

Example: deployment on multicores

<table>
<thead>
<tr>
<th></th>
<th>Min Cores</th>
<th>Partition Size [GB]</th>
<th>RT [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Nehalem</td>
<td>2</td>
<td>4</td>
<td>6.54</td>
</tr>
<tr>
<td>AMD Barcelona</td>
<td>5</td>
<td>1.6</td>
<td>3.55</td>
</tr>
<tr>
<td>AMD Shanghai</td>
<td>3</td>
<td>2.6</td>
<td>4.33</td>
</tr>
<tr>
<td>AMD MagnyCours</td>
<td>2</td>
<td>2</td>
<td>7.37</td>
</tr>
</tbody>
</table>

Experiment setup
- 8GB datastore size
- SLA latency requirement 8s
- 4 different machines

Jana Giceva, Tudor-Ioan Salomie, Adrian Schüpbach, Gustavo Alonso, Timothy Roscoe: COD: Database / Operating System Co-Design. CIDR 2013
COD: Database/Operating System co-design

What is the knowledge we have?

Who knows what?

Insert interface here

DBMS

Application requirements and characteristics

Hardware & architecture

+ System state and utilization of resources

OS
Cod’s Interface supports

DB storage engine

Policy Engine

Push application-specific facts:
- #Requests (in a batch)
- Datastore size (#Tuples, and TupleSize)
- SLA response time requirement

Needed for:
- cost / utility functions:

\[ RT[ms] = c \times \frac{\#tuples}{\#cores} \times (a \times \#queries + b) \]
COD’s key features

Declarative interface
- Resource allocation for imperative requests
- Resource allocation based on cost functions

Proactive interface
- Inform of system state
- Request releasing of resources
- Recommend reallocation of resources
Experimental results

Adaptability to dynamic system state

Adaptability – Latency

Latency [sec]

SLA

Elapsed time [min]

Experiment setup

- AMD MagnyCours
- 4 x 2.2GHz AMD Opteron 6174 processors
- Total Datastore size 53GB
- Noise: other CPU-intensive threads spawned every 4-5min on core 0
Experimental results
Adaptability to dynamic system state

Adaptability – Latency

Latency [sec]

Elapsed time [min]

Naïve datastore engine
SLA

Experiment setup
- AMD MagnyCours
- 4 x 2.2GHz AMD Opteron 6174 processors
- total Datastore size 53GB
- Noise: other CPU-intensive threads spawned every 4-5min on core 0
Experimental results

Adaptability to dynamic system state

Adaptability – Latency

- Naïve datastore engine
- SLA
- COD

Elapsed time [min]

Latency [sec]

Experiment setup

- AMD MagnyCours
- 4 x 2.2GHz AMD Opteron 6174 processors
- total Datastore size 53GB
- Noise: other CPU-intensive threads spawned every 4-5min on core 0
The case for sharing
Pipeline parallelism

- SharedDB does not run queries individually (each one in one thread). Instead, it runs operators that process queries in batches thousands of queries at a time.

Shared DB can run TPC-W!
For the non-db people

- TPC-W has updates!!!
- Full consistency without conventional transaction manager
- Transactions are no longer what you read in textbooks ...
  - Sequential execution
  - Memory CoW (Hyder, TU Munich)
  - Snapshot isolation
Raw performance
Predictability, robustness

Graph: Load Interaction between Light and Heavy Queries

- Small queries
- Load
- MySQL
- SystemX
- SharedDB

Throughput (Web Interactions/second) vs. Percentage of Heavy Queries in the Workload.
It is the data, stupid
Not everything is parallel

P. Roy, J. Teubner, G. Alonso
Efficient Frequent Item Counting in Multi-Core Hardware, KDD 2012
The data ties it all together

- The previous example makes a case for all the ideas described
  - Hardware acceleration on the data path
  - Knowing where to do what
  - In network data filtering
  - On the fly statistics
  - Characterizing the hardware and the load
Conclusions
The opportunity is now

- Consensus on major crisis in hardware (from the sw perspective)
- Hardware not really improving, responsibility passed on to software
- Business models and IT systems moving towards specialization
  - Room for customized systems
  - Need for general solutions