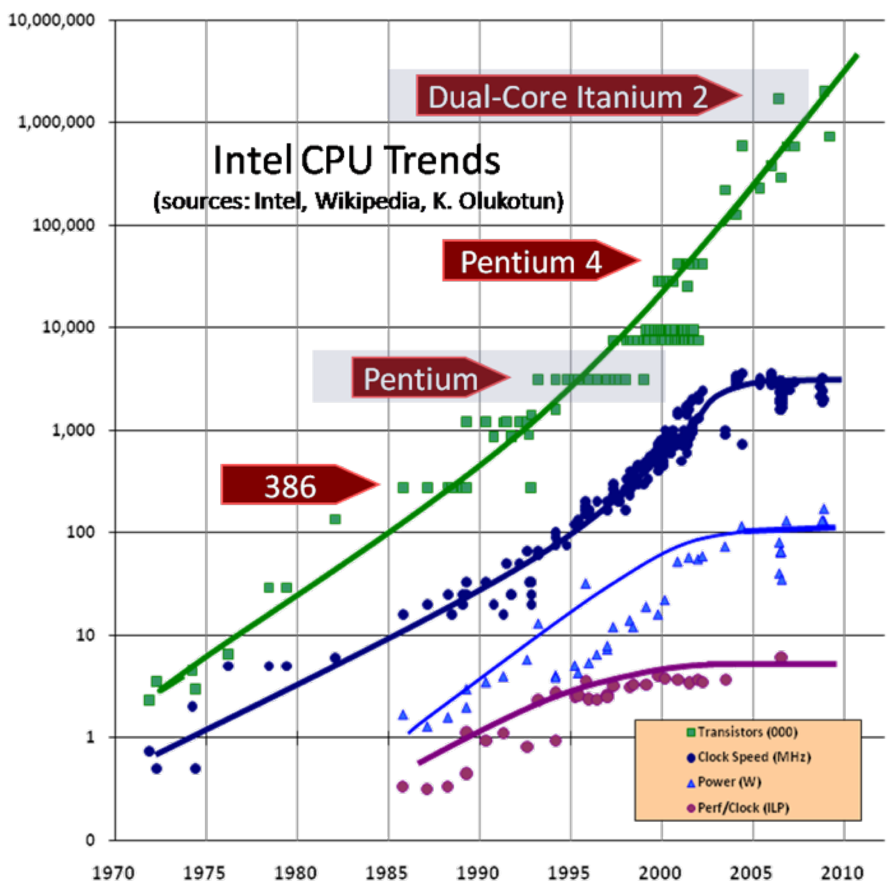
In-Memory Data Management Research

Martin Boissier, Markus Dreseler

October 2015

“The Free Lunch Is Over”



- Number of transistors per CPU increases
- Clock frequency stalls

[Source: http://www.gotw.ca/publications/concurrency-ddj.htm]

Capacity vs. Speed (latency)

- Memory hierarchy:
  - Capacity restricted by price/performance
  - SRAM vs. DRAM (refreshing needed every 64ms)
  - SRAM is very fast but very expensive

➡ **Memory is organized in hierarchies**

- Fast but small memory on the top
- Slow but lots of memory at the bottom

	technology	**latency**	**size**
CPU	SRAM	< 1 ns	bytes
L1 Cache	SRAM	~ 1 ns	KB
L2 Cache	SRAM	< 10 ns	MB
Main Memory	DRAM	100 ns	GB
Magnetic Disk		**~ 10 000 000 ns** (10 ms)	**TB**

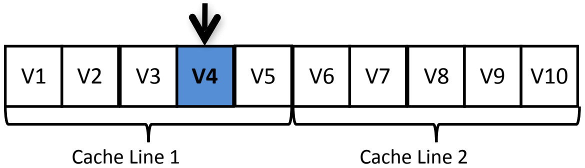
Data Processing

- In DBMS, on disk as well as in memory, data processing is often:
  - Not CPU bound
  - But bandwidth bound
  - "I/O Bottleneck"

➡ CPU could process data faster

- Memory Access:
  - **Not** truly random (in the sense of constant latency)
  - Data is read in blocks/cache lines
  - Even if only parts of a block are requested

➡ Potential waste of bandwidth



Memory Hierarchy

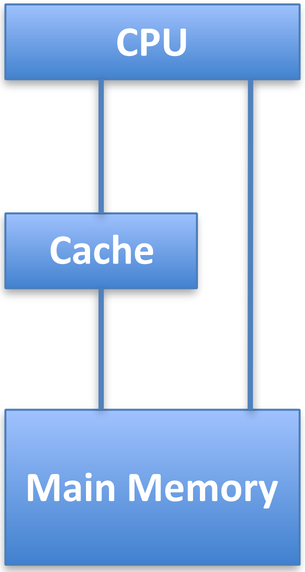
- **Cache**
  Small but fast memory, which keeps data from main memory for fast access.

➡ **Cache performance is crucial**

- Similar to disk cache (e.g. buffer pool)

But: Caches are controlled by hardware.

- **Cache hit**
  Data was found in the cache.
  Fastest data access since no lower level is involved.
- **Cache miss**
  Data was not found in the cache. CPU has to load data from main memory into cache (miss penalty).



Locality is King!

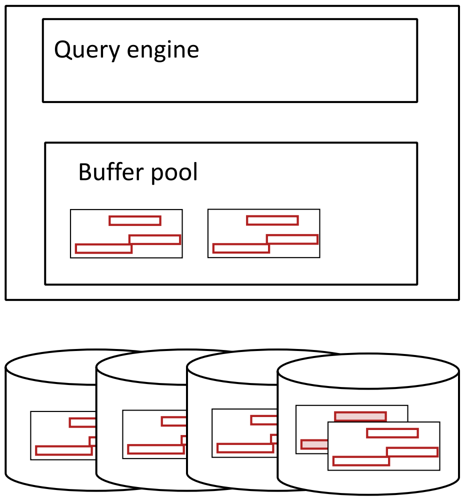
- To improve cache behavior
  - Increase cache capacity
  - Exploit locality
    - Spatial: related data is close (nearby references are likely)
    - Temporal: Re-use of data (repeat reference is likely)
- To improve locality
  - Non random access (e.g. scan, index traversal):
    - Leverage sequential access patterns
    - Clustering data to a cache lines
    - Partition to avoid cache line pollution (e.g. vertical decomposition)
    - Squeeze more operations/information into a cache line
  - Random access (e.g., hash joins):
    - Partition to fit in cache (cache-sized hash tables)

Motivation

- Hardware has changed
  - TB of main memory are available
  - Cache sizes increased
  - Multi-core CPU's are present
  - Memory bottleneck increased
  - NUMA (and NUMA on a NUMA?)
- Data / Workload
  - Tables are wide and sparse
  - Lots of set processing
- Traditional databases
  - Optimized for write-intensive workloads
  - Show bad L2 cache behavior

Problem Statement

- DBMS architecture has **not changed** over decades
- Redesign needed to handle the changes in:
    - Hardware trends (CPU/cache/memory)
    - Changed workload requirements
    - Data characteristics
    - Data amount

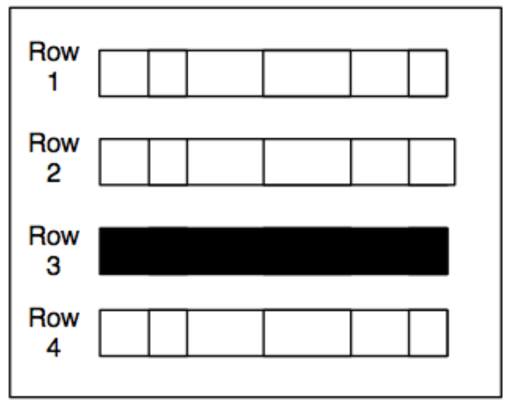


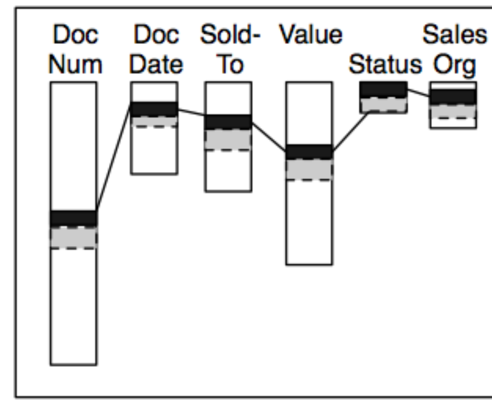
Traditional DBMS Architecture

Row- or Column-oriented Storage

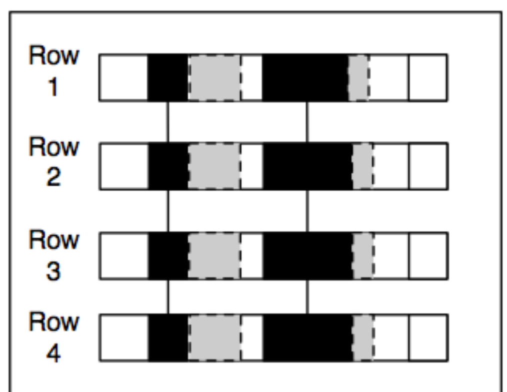
Row Store | Column Store

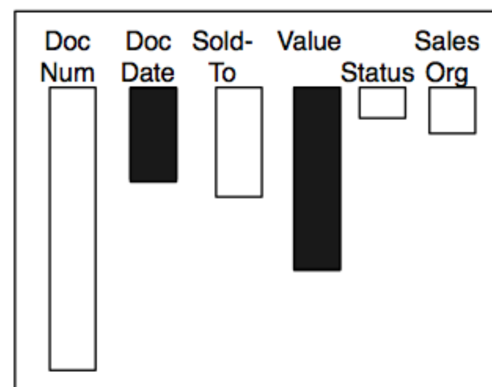
SELECT *
FROM Sales Orders
WHERE Document Number = ‘95779216’





SELECT SUM(Order Value)
FROM Sales Orders
WHERE Document Date > 2009-01-20





Question & Answer

- How to optimize an IMDB?
  - Exploit sequential access, leverage locality

    -> Column store
  - Reduce I/O
    - Compression
  - Direct value access

    -> Fixed-length (compression schemes)
  - Late Materialization
  - Parallelize

Seminar Organization

Objective of the Seminar

- Work on advanced database topics in the context of in-memory databases (IMDB) with regards to enterprise data management
- Learn how to work scientifically
  - Fully understand your topic and define the objectives of your work
  - Propose a contribution in the area of your topic
  - Quantitatively demonstrate the superiority of your solution
  - Compare your work to existing related work
  - Write down your contribution so that others can understand and reproduce your results

Seminar schedule

- Today (15.10.): Overview of topics, general introduction
- Thursday (27.10.): In-memory DB Basics & HYRISE (if you're interested)

- 22.10.: Send your priorities for topics to *martin.boissier@hpi.de*

- Planned Schedule
  - **15./17.12.2015:** Mid-term presentation
  - **16./18.02.2016:** Final presentation (tbc)
  - **29.02.2016:** Peer Reviewing (tbc)
  - **20.03.2016:** Paper hand-in (tbc)

- Throughout the seminar: individual coaching by teaching staff
- Meetings (Room V-2.16)

Final Presentation

- Why a final presentation?
  - Show your ideas and their relevance to others
  - Explain your starting point and how you evolved your idea /implementation
  - Present your implementation, explain your implementations properties
  - Sell your contribution! Why does your work qualify as rocket science?

Peer Reviewing

- Each student will be assigned a colleague's paper version (~2 weeks before paper hand-in)
  - Review will be graded
  - Annotate PDF for easy fixes (e.g., typos)
  - Short summary (2-3 pages in Word) about the paper's content and notes to the author how to further improve his paper
- Expected to be done in the week from February 29 to March 4

Final Documentation

- 7-9 pages, IEEE format [1]
- Suggested Content: Abstract, Introduction into the topic, Related work, Implementation, Experiment/ Results, Interpretation, Future Work
- Important!
  - Related work needs to be cited
  - Quantify your ideas / solutions with measurements
  - All experiments need to be reproducible (code, input data) and the raw data to the experiment results must be provided

[1] http://www.ieee.org/conferences_events/conferences/publishing/templates.html

Grading

- 6 ECTS
- Grading:
  - 30% Presentations (Mid-term 10% / Final 20%)
  - 30% Results
  - 30% Written documentation (Paper)
  - 10% Peer Review

Topic Assignment

- Each participant sends list of top three topics in order of preference to lecturers by 22.10.
- Topics are assigned based on preferences and skills by 26.10.

HYRISE

- Open source IMDB
- Hosted at https://github.com/hyrise
- C++11
- Query Interface: Query plan or stored procedures

Recommended Papers for Intro

- Plattner, VLDB 2014: *The Impact of Columnar In-Memory Databases on Enterprise Systems*
- Grund et al. VLDB 2010: *HYRISE—A Main Memory Hybrid Storage Engine*
- Krueger et al. VLDB 2012: *Fast Updates on Read-Optimized Databases Using Multi-Core CPUs*

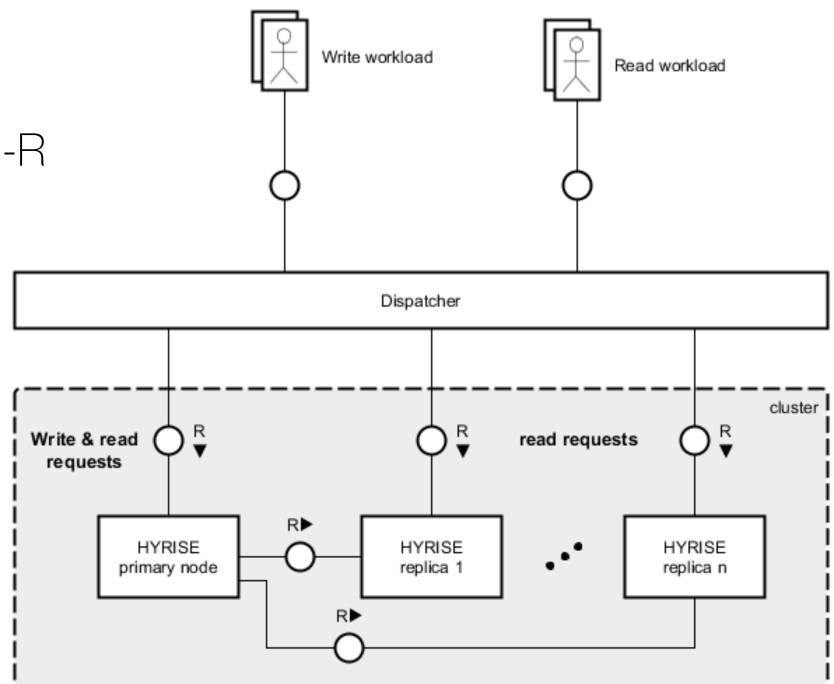
Topics

Topics

- In-Memory Performance & HYRISE
  - HYRISE topics (indices, NVRAM, replication)
  - SGI topics (cache coherence)
  - Co-processing (GPU and Xeon Phi)
- Workload analyses & benchmarking
  - Performance evaluations (relational vs. k/v)
  - Analyzing synthetic benchmarks (TPC-C/E)

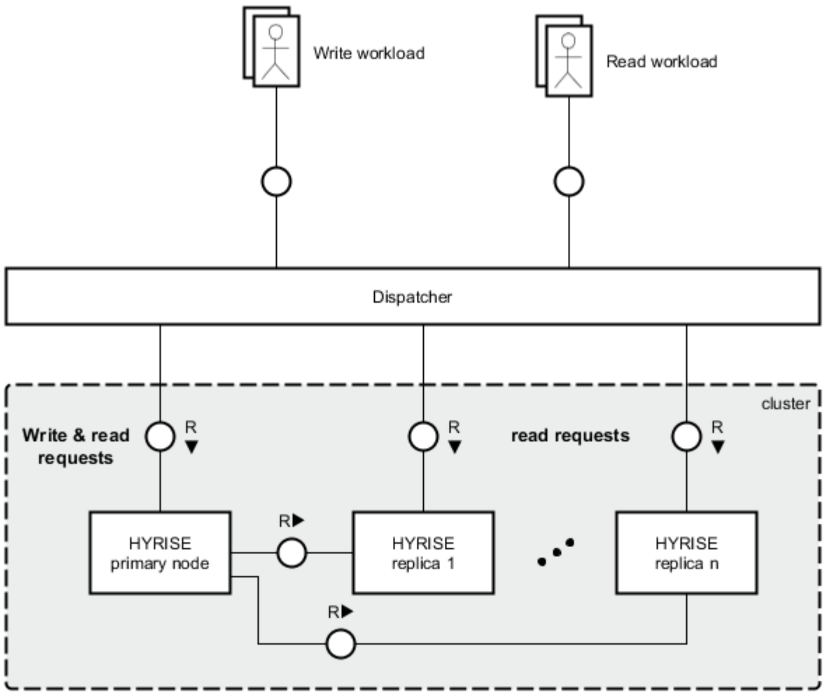
K-Safety in Hyrise-R

- Project
  - Hyrise-R - Scale-Out and Hot-Standby version of Hyrise
  - Hyrise-R implements Lazy Master Replication
- Tasks
  - Evaluate and implement K-Safety for Hyrise-R
  - Demo scenario
  - Performance evaluation
- Technologies
  - Hyrise
  - C/C++



Elasticity in Hyrise-R

- Project
  - Hyrise-R - Scale-Out and Hot-Standby version of Hyrise
  - Hyrise-R implements Lazy Master Replication
- Tasks
  - Implement Elasticity for Hyrise-R
  - Demo scenario
  - Smart query distribution
    - Different indices
    - Different latencies in federated cloud
- Technologies
  - Hyrise
  - C/C++
  - Docker



Detection of compound events in spatio-temporal football data

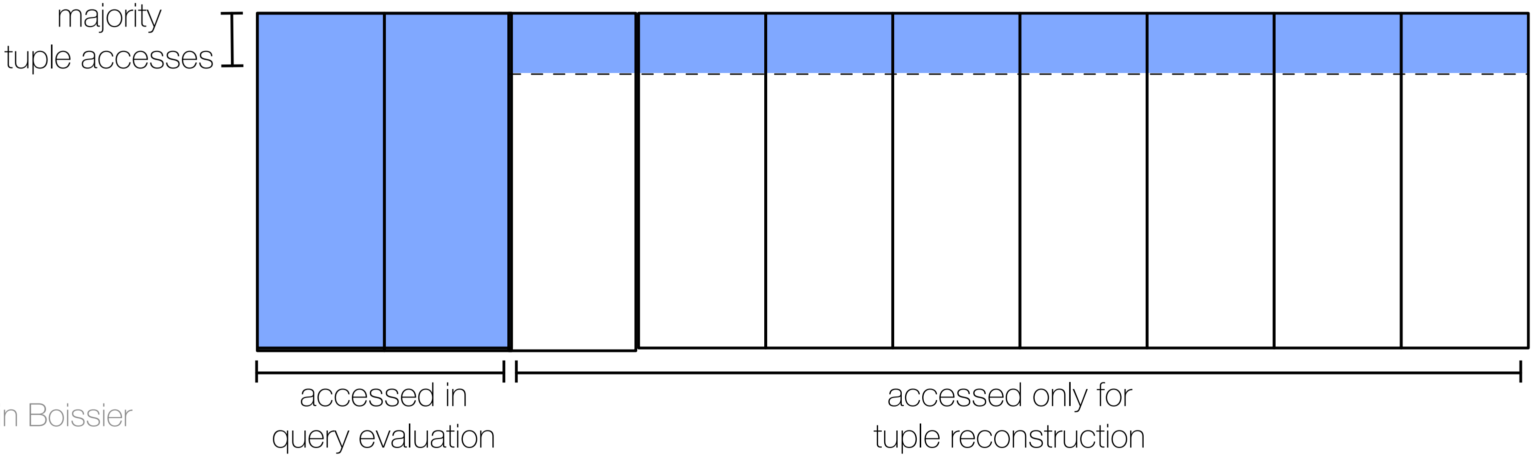
- **Project:**
  - The usage of spatial-temporal data increased strongly in recent years (e.g. performance analytics in sports)
  - Provided data for football games of the German Bundesliga
    - 1.5 million position information per game
    - Manually tracked event list
  - Problem: the event list is tracked manually, is not synchronized with the position information, and contains errors
- **Goal:**
  - *Implementation and evaluation of algorithms to automatically detect compound events in positional data of football games*
  - *Leveraging the parallel computation capabilities of coprocessors*

Data Tiering / Aging

- **Data Aging:** within its life time, data usually loses relevance and can be stored more price-efficient
- **Data Tiering:** storing (evicting) data on different storage tiers based on their access frequency / relevance
  - Classical databases' *caching*: hot data is cache in DRAM
  - Modern main memory databases (*"anti-caching"*): cold data is moved to secondary storage

Simplified Data Tiering for HYRISE

- **Project:**
  - Data Tiering can be transparently handled by APIs, which tier data based on a given temperature
  - The idea: while retaining the performance superiority of IMDBs, find columns that are never scanned and only accessed for point-accesses
  - These cold columns are moved out of DRAM

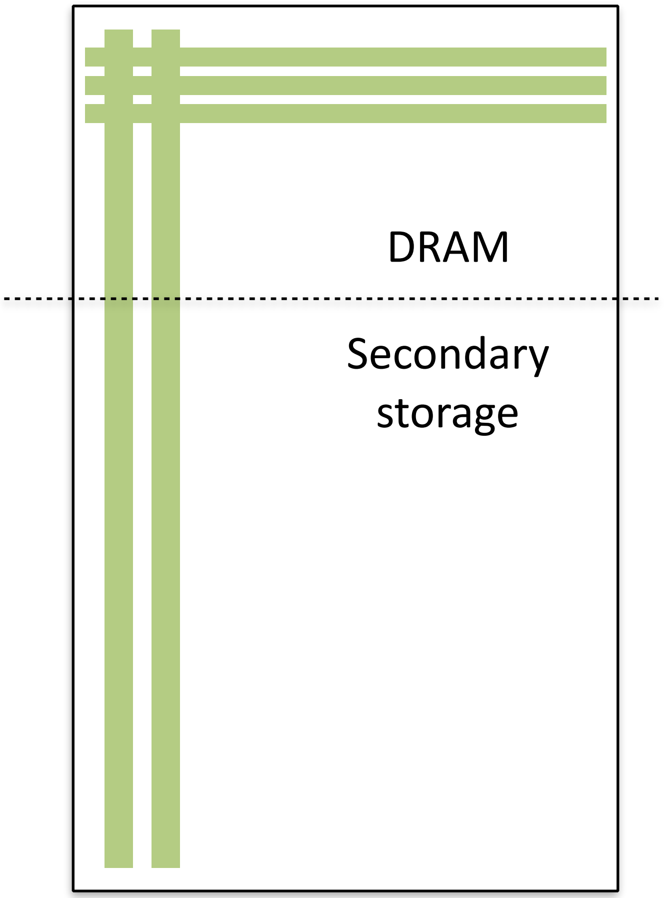


Simplified Data Eviction for HYRISE

- Tasks:
  - Set data temperatures based on expected accesses
  - Storage “drives”:
    - emulated RAM-disk with adjustable characteristics
    - top-notch PCI-e NAND Flash with 6+ GB/s
- Goal:
  - *Evaluate and implement data tiering for HYRISE*
  - *Measure performance for industry-standard benchmark TPC-C*

How to Sort a Table

- Project:
  - SAP HANA can partition tables into a hot partition and a cold partition (on SCM)
  - Given an SQL workload, sort a table optimally to gather full-width accesses in the hot partition
- Goal:
  - *Evaluation of a optimal sorting approach on SAP HANA for a real enterprise workload*



An NVRAM-emulating Allocator

- **Project:**
  - NVRAM is coming, but there is no hardware yet
  - Emulation is complicated and requires specialized hardware
- **Goal:**
  - *Exploit NUMA latencies in order to allocate memory (e.g., using move_page) on a distant NUMA node, which has similar latency/bandwidth characteristics*

Socket	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	104,6	190,4	221	196,3	480,9	485	488,8	494	490,9	493,1	490,2	495,5	490	493,4	490,6	493
1	184,9	105,4	185,1	219,5	480,5	483,8	488,1	492,9	489,8	491,9	489,6	494,4	489	492,3	489,5	492
2	215,2	189,1	105,5	188,3	489,5	494	481,3	487,6	490,9	493,3	490,2	495,6	491	494,7	491,4	493,3
3	192,5	219,9	184,3	104,5	489	492,5	480,5	485,2	489,5	491,8	488,8	494,6	489,8	493	490,6	491,8
4	481,8	484,1	489,9	493,6	105,4	188,8	217,3	193,8	481,5	482,5	489,4	494,1	489,6	494	490,8	492,4
5	481,6	484	489,9	493,6	184,3	105	185,2	220,3	481,4	482,5	489,2	495	489,5	493,6	490,1	492,6
6	490	493	481,7	484,8	216,8	188,8	106	187,8	490,3	492,1	480,1	485,8	489,1	493,7	490,9	493,1
7	491,8	494,1	483,1	486	190,9	218,6	183,5	105,9	491,1	493	481,6	487,5	489,8	494,7	492,7	494,5
8	492,6	495,3	492,3	495,6	481,5	486	491	495,5	106	188,1	217,9	193,7	482,6	488,4	493,2	494,9
9	490,3	493	490,9	493,1	480	483,5	489,4	493,6	184,8	106,5	185,6	219,4	481,7	485,3	491,1	492,7
10	490,5	493,5	490,6	494,4	489,2	492,8	480,9	485,2	217,1	189	104,9	187,4	489,5	494,3	482,4	483,9
11	492,5	495,9	492,6	496	491,1	494,9	482,2	488,3	191,8	219,1	184,4	105,6	491,6	496,5	482,8	486,2
12	490,3	494	491,8	495,6	490	494,2	489,4	494,6	483,1	484,5	490	496,2	105,2	187,8	217,7	193,4
13	491,2	493,8	491,3	495,7	489,9	494,1	490,4	494,6	482,9	484,6	489,8	495,9	184,8	105,3	185,7	219,1
14	491,5	494,1	490,4	494,9	489,6	494,2	491,1	495,3	491,5	493,5	481,9	487,5	216,2	188,5	105,8	187,1
15	491,3	494	490,8	495,2	490	494,4	490,2	495,5	491,3	493,2	481,8	487,5	189,8	218,6	185	105,8

Relaxed Cache Coherence

- **Cache Coherence on the SGI UV300**
  - Processors need to keep CPU caches coherent with the memory
  - When two processors access the same address, they need to see the same value
  - Ensuring cache coherence is expensive, more expensive across NUMA nodes, and even more so across blades
  - Can we improve performance by selectively working around coherence protocols?

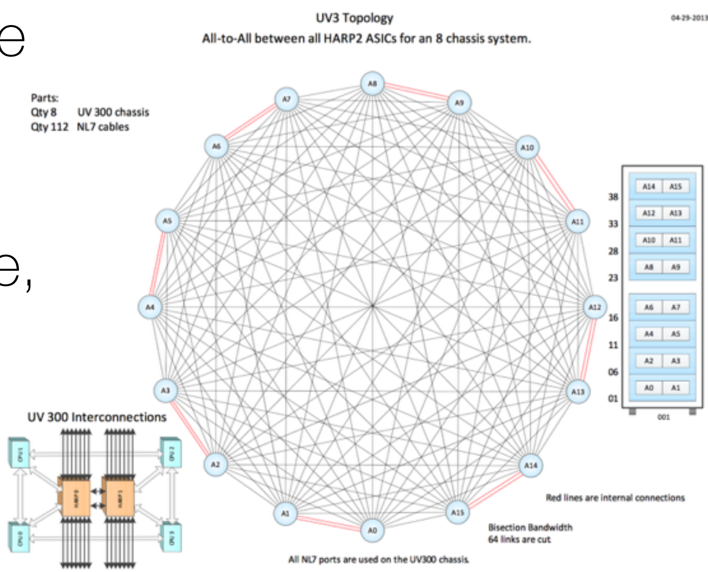


Figure 3-5 Single Rack Topology

Relaxed Cache Coherence

- Setup
  - We have an SGI machine with 480 logical cores and 12 TB DRAM
  - SGI is very interested and will provide support for the project

- Tasks
  - *Step 1*: in micro benchmarks, identify the cost of coherence
  - *Step 2*: measure performance costs in HYRISE and identify potential points for snapshot coherence
  - *Step 3*: implement optimizations and benchmark

- Prerequisites
  - this project requires solid C++ knowledge

TPC-DS on HYRISE

- Project
  - TPC-DS is a well-known benchmark in the area of decision support
  - Read-only: only selects are performed, no updates
  - Queries are long-running and complex

- Tasks
  - *Step 1*: for a selected number of queries, write JSON queries for HYRISE
  - *Step 2*: implement needed operators, such as `IN`
  - *Step 3*: optimize performance of query plans

In-Memory Database Coprocessing

- Status Quo:
  - Application logic moves closer to the database layer
  - Compute intensive, long running application transactions consume computational power of the database system
  - Classical database tasks have less available resources

- Solution:
  - Coprocessors like Nvidia's Tesla or Intel's Xeon Phi can be used to increase the amount of computational power for specific application logic within the database system

In-Memory Database Coprocessing Application Example

- Application Example:
  Product Cost Calculation

- Logic can be expressed as system of linear equations
- Matrix inversion and matrix vector operations can be used to solve the problem efficient on coprocessors

pc →Product cost per unit
pp →Purchase price for material
mc →Manufacturing costs for material
b_{ij} →Bill of material: Number of units of product i required to produce one unit of product j
a_{ij} → Activity required to produce one unit of j
r_i → Cost center rate of cost center i
l_j → Capacity load of cost center j
p_i → Primary demand of product i
s_j → Secondary demand of product j

Raw Materials:

$$pc = pp$$

(Semi) Finished Goods:

$$pc_j = mc_j + \sum_i b_{ij} pc_i$$

Manufacturing Costs:

$$mc_j = \sum_i a_{ij} r_j + \sum_i b_{ij} mc_i$$

Secondary Demand:

$$s_j = \sum_i b_{ij}(p_i + si)$$

Capacity load:

$$l_j = \sum_i a_{ij}(p_i + si)$$

$$s = B(p+s) \qquad t - p = Bt$$

$$l = A(p+s) \qquad l = At$$

$$\begin{pmatrix} p \\ 0 \end{pmatrix} = \begin{pmatrix} 1-B & 0 \\ A & -1 \end{pmatrix} \begin{pmatrix} t \\ l \end{pmatrix}$$

$$t_i = s_i + p_i$$

$$\sum_i mc_i(p_i + s_i) = \sum_{ij} b_{ji} mc_j (p_i + s_i) + \sum_i pp_i (p_i + s_i) \quad (7)$$

$$\sum_i mc_i s_i = \sum_{ij} b_{ij} mc_i (p_j + s_j) \quad (8)$$

$$\sum_i mc_i p_i = \sum_i pp_i (p_i + s_i) \quad (10)$$

$$pp = mc - B^T mc - A^T r$$

$$pr = Lr \text{ with } L_{ij} = \delta_{ij}(\sum_k c_{jk} + l_j) - c_{ji}$$

$$pr_i = pp_i t_i, \; r_i = mc_i, \; c_{ij} = b_{ij} t_j$$

$$mc_i = \sum_j b_{ji} mc_j + pp_i \quad (6)$$

$$mc_{ik} = (1 + z_i)(\sum_j b_{ji} mc_{jk} + pp_i)$$

$$\sum_i mc_i p_i = \sum_i pp_i (p_i + s_i) + \sum_i pf_i \quad (14)$$

$$mc_i = (1 + z_i)(\sum_j b_{ji} mc_j + pp_i) \quad (11)$$

$$s_i = \sum_j (b_{ij}(p_j + s_j) + f_{ij}) \quad (12)$$

$$t_i mc_i = \sum_j (t_i b_{ji} + f_{ji}) mc_j + t_i pp_i + pf_i \quad (13)$$

$$cm_{ij} = (p_{ij}\, sp_{ij} - p_{ij}\, sp_{ij}\, sdp_{ij} - p_{ij}\, sdr_{ij}) fx_j - p_{ij}\, mc_i$$

In-Memory Database Coprocessing Topic: Database Throughput

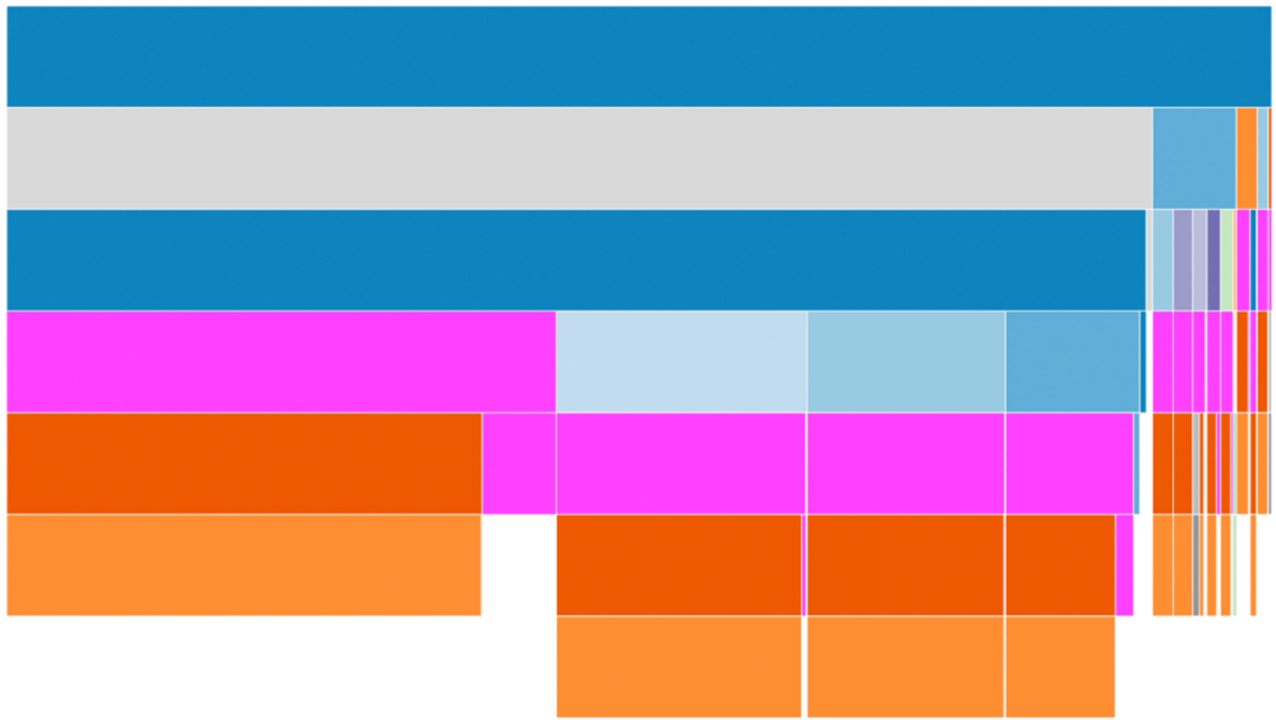
- Hypothesis:
    - Coprocessors increase the throughput within an application logic enriched database system
- Goal:
    - Improve an existing benchmark to prove the hypothesis

In-Memory Database Coprocessing Topic: Application Performance (1/2)

Product Cost Simulation Performance Analysis

Style: Icicle
Depth: 10
Cutoff: None

Operation:
database



Search: database

Number of Executions	SUM(time) in seconds	AVG(time) in seconds	call
165	63.876	0.387	/bill_of_material/json_materialization /database
165	55.271	0.335	/bill_of_material/json_materialization /database/select
165	55.171	0.334	/bill_of_material/json_materialization /database/select/nanodbc::execute

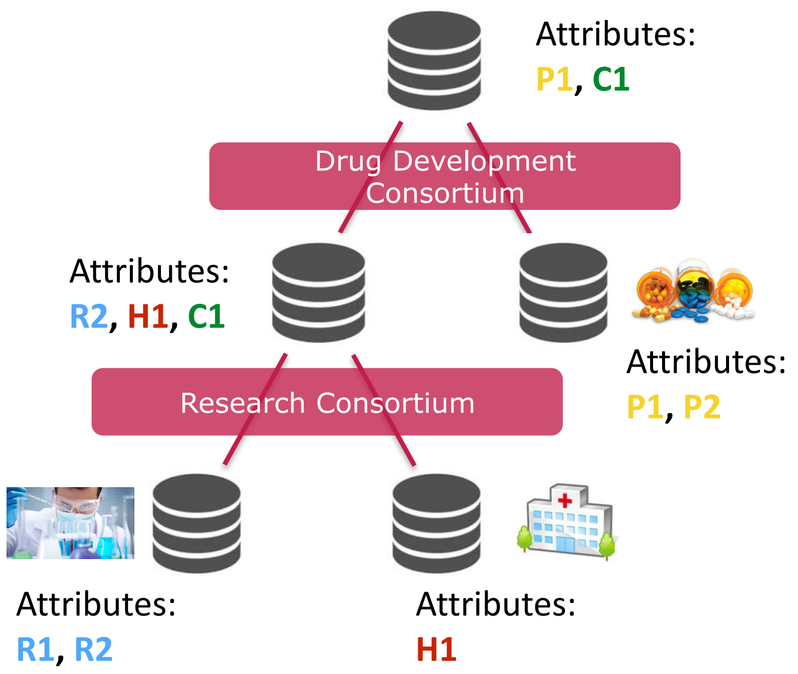
In-Memory Database Coprocessing Topic: Application Performance (2/2)

- Hypothesis:
  - Placing the database directly on the coprocessor improves application performance significantly

- Goal:
  - Replace database layer used by the current prototype, leveraging a coprocessor database
  - Improve the overall performance characteristics of the application

A Federated In-Memory Database for Life Sciences

- Tasks
  - Connect distributed databases to form a federated in-memory database system
  - Benchmark
    - Sensitive data must reside locally
    - Minimal data exchange between nodes
    - Fast and complete query propagation
- Goal
  - Provide an uniform interface to enable distributed analysis of heterogeneous medical data
- Technologies
  - HANA
  - C++



Workload Analyses & Benchmarking

Synthetic Benchmarks: *How synthetic are they really?*

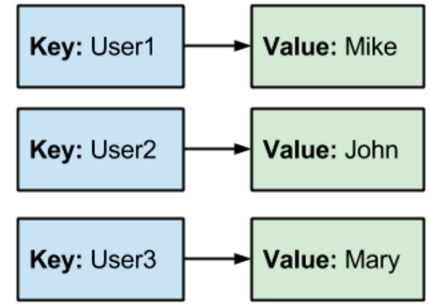
- Project:
  - Plattner et al. found that benchmarks do not reflect the properties of real-world systems
  - Many research paper make assumptions that do not reflect real systems in any way
  - *How do both synthetic benchmarks compare to real enterprise systems?*

[2] Plattner, The Impact of Columnar In-Memory Databases on Enterprise Systems, VLDB 2014

Synthetic Benchmarks: *How synthetic are they really?*

- Tasks:
    - Trace and analyze synthetic benchmarks
    - Determine and evaluate relevant characteristics for database performance (index usage, expensive stateements, …)
    - Analyze each workload thoroughly and compare with traces of a real ERP system
- Goal:
    - *Thorough comparison of a productive enterprise system with synthetical enterprise benchmarks*

Suitability of an EAV model in IMDBs



Enterprise systems are diverse

Non-functional system characteristics, e.g., flexibility, performance, impact the data model design

Example: medical information systems often use an entity-attribute-value (EAV) model

- » How does that work with an columnar IMDB?
- » Would it make sense to use that in other domains as well?
- » …

Suitability of an EAV model in IMDBs

- Tasks
  - Implementation of key-value (KV)/EAV data model for KV Store and IMDB
  - Benchmark
    - Performance for different query classes
    - Memory consumption

- Goal
  - Comparison of differences WRT performance and memory usage
  - Identification of query classes that perform comparatively poor on KV stores

- Technologies
  - HANA
  - Cassandra (Scylla)

Thank you.