Emergent Hardware for Databases with focus on Semantic Web: Semantic Hybrid Multi-model Multi-platform (HM3P) Databases

Professor Dr. rer. nat. habil. Sven Groppe
https://www.ifis.uni-luebeck.de/index.php?id=groppe
Agenda: Types of Database Management Systems (DBMS)

- Cloud DBMS
- Hardware-Accelerated DBMS (GPU, FPGA, Quantum)
- IoT DBMS
- Mobile DBMS
- Federated DBMS
- Multi-Model DBMS
  - relational, XML, JSON, graph, Semantic Web, unstructured
- Multi-Platform DBMS
  - Examples
  - Multi-Platform Development
- Hybrid Multi-Model Multi-Platform (HM3P) DBMS
  - Challenges
Zoo of **Data Formats**, for example:

- **relational data**  
  - in relational databases
- **XML**  
  - for exchange
- **JSON**  
  - web data
  - Semantic Web
- **graph data**  
  - from social networks
- **unstructured data**  
  - of social media like wikis

➤ Parallel use of different **Data Models** for storing and processing
Every data model (here Semantic Web) has its own set of languages (data, query, rule, ...)

Semantic Web (Core) "Standards"

- Query: SPARQL
- Ontology: RDFS, OWL (2)
- Rule: RIF
- Data Format: RDF
Semantic Web: Ontology

- **Ontology as additional abstraction layer**
  - More than schema descriptions:
    - Specification of background knowledge (based on which new facts can be derived)
      - avoids storing of redundant data
      - supports re-use of data
      - supports data integration
      - increases computational complexity
Special Concepts 1/2: Open world assumption (OWA)

- **Closed World Assumption (CWA) in Databases:**

  "The database contains all and anything not contained in the database is presumed to be false/not existent!"

- **Open Context like Web**
  ➔ CWA is false!
Special Concepts 1/2: Open world assumption (OWA): Example

- Data source 1 contains:
  "There exists a flight at 2pm"
  "There exists a flight at 3pm"
- My query:
  "Is there a flight at 5pm?"
- CWA Result: No!
- OWA Result: unknown!
  - i.e., there could be a data source 2, which contains the information about a flight at 5pm!
  Data source 2 is maybe currently not integrated or currently not available...
Special Concepts 2/2: **No unique name assumption**

**Example:**
A child has two parents: (in DL: $\text{Person} \sqsubseteq \leq 2 \text{hasParent\_Person}$), but the following facts seem to be conflicting:
Special Concepts 2/2: No unique name assumption

Example:
A child has two parents: (in DL: `Person \sqsubseteq 2 \text{hasParent}.Person`), but the following facts seem to be conflicting:

No unique names/keys
⇒ JohnDoe, JohnMine and LisaMine are not necessarily different objects (here persons)
Special Concepts 2/2: No unique name assumption

4 possibilities:

1. JohnDoe ≡ JohnMine
2. JohnDoe ≡ LisaMine
3. JohnMine ≡ LisaMine
4. JohnDoe ≡ JohnMine ≡ LisaMine
Special Concepts 2/2: No unique name assumption

4 possibilities:

1. **JohnDoe** \equiv JohnMine
2. JohnDoe \equiv LisaMine
3. JohnMine \equiv LisaMine
4. JohnDoe \equiv JohnMine \equiv LisaMine

Only 1. is intuitive for humans!

Adding following facts and axioms:

- Woman \underbrace{\text{type}}_{\text{disjoint with}} \text{Man}
- LisaMinelly
- JohnDoe
- JohnMinelly

⇒ automatic inference of 1. possibility!
Semantic Web DBMS **LUPOSDATE**

**Support of:**
- SPARQL Queries
- RIF Rules
- RDF Schema
- OWL (via OWL2RL in RIF)

**Indexing:**
- Stream Processing
- Main memory for small datasets
  - RDF3X
- Disk-based for large datasets
- Cloud: HBase
- P2P

**Visualizations:**
- Visual Editor
  - Queries (SPARQL)
  - Rules (RIF)
  - Data (RDF) in
    - 2D and
    - 3D
  - Logical Optimization Rules
- Summaries of RDF Data
- Operator graph
- Processing of Queries and Rules
- Optimization Steps

**Extra:**
- Parallel Processing
- Distributed Processing
- Cloud Computing
- Mobile Computing
- P2P for Internet of Things
- Compression of RDF Data
- Embedding of SW Languages in Programming Languages
- Speeding up by FPGAs
**RDF3X - Indexing Scheme for large-scale RDF triple stores**

Prefix-Search in **Index** with (Prefix-)**Key**:
- **PSO**
  - rdf:type ont:Car
- **SPO**
  - data:Sven ont:Owns ?car

Result of Triple Pattern:
- **?car**
  - HL 100
  - HL 101
  - ...
  - HL 999
  - ...

Search in **PSO** – **B***-tree:
- search(rdf:type ont:Car)
  - next(?car ≥ HL 999)
  - next(?car ≥ HL 999)
  - ... Heuristics: continue search in inner nodes if not found in next leaf node!

**Merge**(?car)
- Sorted by ?car

Maximise usage of merge joins
⇒ 3!=6 indices for all possible sort orders of triples SPO:
  - SPO, SOP, PSO, POS, OSP, OPS

**Complexity of Merge Join** ∆,**Merge**:
- Worst Case (duplicates): \( O(|R| \times |S|) \)
- without duplicates: \( O(|R| + |S|) \)
- with sideways information passing: \( O(|R| \triangleleft |S|) \)
  (assuming quasi-constant access in B*-tree)
Polyglot Persistence

- data sources: integration at application level
- performance of data processing cannot be fully optimized
- fault-tolerance cannot be transparently offered across the different databases
- zoo of query languages
+ features of different types of databases can be used

Multi-Model DBMS (MM-DBMS)

+ full and uniform data integration at database level
+ performance: fully optimized across different data models
+ transparent fault-tolerance
+ SQL standards: relational ('87), XML ('03), temporal ('11), JSON ('16), Multi-dimensional Arrays ('19), schemaless ('19), streams ('20?), property graphs ('21?)
- features of different types of databases cannot be used
Federated DBMS

- Bottom-up-integration of existent databases
- mostly independent DBMS with private conceptual database schemes
- partially enabling external accesses (in cooperation)
- heterogeneity of data models and transaction management possible (but relational DBMS in most times)

- problems with semantic heterogeneity
- transparency in distribution only partially achievable
One Size-Approach

- M. Stonebraker, U. Cetintemel. "One Size Fits All": An Idea Whose Time Has Come and Gone.
  ICDE 2005
  - The last 25 years of commercial DBMS development can be summed up in a single phrase: "One size fits all".
  - ...this concept is no longer applicable to the database market...

- Our approach: **Enlarge the size!**
  - Over the boundaries and limitations of single platforms and their specialized approaches
  - Increase transparency, performance and ease of use
Hybrid Multi-Model Multi-Platform (HM3P) Database

- **full and uniform data integration** at database level
- **performance**: fully optimized across different data models
- **transparent fault-tolerance**
- **SQL standards**: relational ('87), XML ('03), temporal ('11), JSON ('16), Multi-dimensional Arrays ('19), schemaless ('19), streams ('20?), property graphs ('21?)

**features of different types of databases running on different platforms** can be used
**Variant: Semantic HM3P (SHM3P) DB**

Single instance of **SHM3P Database** offers (fully cross-platform optimized) functionality of & replaces

- **IoT DB On the Edge**
  - Lightweight reasoning on large data sizes of IoT devices

- **Main-Memory DB**
  - Heavyweight reasoning on moderate data sizes

- **Quantum DB**
  - Heavyweight reasoning on large data sizes

- **Cloud DB**
  - Reasoning on small data sizes of mobile devices

**Reasoning:**

- **How to integrate the different reasoning capabilities and requirements into one transparent global reasoner?**

  - **Semantic Layer as glue** between other models and platforms
  - **new challenges** like integrating different types of reasoners in a transparent global reasoner

  - **Features of HM3P databases**
  - **Easier data integration**
  - **Performance issues** may occur due to semantic layer
Types of DBMS

**Model Diversity**

- **One Model**
- **Multi-Model**

**Support of semantic layer**

**Platform Diversity**

- **Single Platform**
- **Multiple Platforms**
- **Hybrid Multiple Platforms**


- State-of-the-art partly/ rudimentarily addressed visionary/ single attempts (e.g. hybrid cloud)
- Global semantic layer without semantic layer
The ark is too small...
The ark is too small...

... but there is always enough space for the own product/research system! → luposdate3000
Platform-specific types of DBMS
## Examples of Multi-Platform Databases 1/2

<table>
<thead>
<tr>
<th>Type</th>
<th>DBMS</th>
<th>Ext.</th>
<th>Models RCKJXGDO</th>
<th>Query Languages</th>
<th>Platforms NJWLUMSZCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>PostgreSQL</td>
<td>I</td>
<td>R-KJX--O</td>
<td>extended SQL</td>
<td>N-WLUMS-CH</td>
</tr>
<tr>
<td></td>
<td>MS SQL Server</td>
<td>I</td>
<td>R--JXG-0</td>
<td>extended SQL</td>
<td>N-WL-----CH</td>
</tr>
<tr>
<td></td>
<td>IBM DB2 LUW</td>
<td>I</td>
<td>R---XGDO</td>
<td>extended SQL/XML</td>
<td>N-WLUS-Ch</td>
</tr>
<tr>
<td></td>
<td>IBM DB2 z/OS</td>
<td>I</td>
<td>R---XGDO</td>
<td>extended SQL/XML</td>
<td>N-------Z--</td>
</tr>
<tr>
<td></td>
<td>Oracle DB</td>
<td>I</td>
<td>R--JX-D0</td>
<td>SQL/XML, SQL/JSON</td>
<td>N-WLUMS*CH</td>
</tr>
<tr>
<td></td>
<td>MySQL</td>
<td>II</td>
<td>R-K----O</td>
<td>SQL, memcached API</td>
<td>N-WLUMS-C-</td>
</tr>
<tr>
<td></td>
<td>Sinew¹</td>
<td>III</td>
<td>R-K-----</td>
<td>SQL</td>
<td>N-WLUMS-CH</td>
</tr>
<tr>
<td>Column</td>
<td>Cassandra</td>
<td>I</td>
<td>-C----G-0</td>
<td>SQL-like CQL</td>
<td>-JWLUMS-CH</td>
</tr>
<tr>
<td></td>
<td>CrateDB</td>
<td>I</td>
<td>RC-J-G--</td>
<td>SQL</td>
<td>-JWL-M--C-</td>
</tr>
<tr>
<td></td>
<td>DynamoDB</td>
<td>I</td>
<td>-CKJ-G-0</td>
<td>simple API (get/put/update)</td>
<td>-JWLUM--C-</td>
</tr>
<tr>
<td></td>
<td>Vertica</td>
<td>II</td>
<td>-C-J-G--</td>
<td>SQL-like</td>
<td>N--LU---CH</td>
</tr>
</tbody>
</table>

**Legend:** Ext.: I = adoption of a new storage strategy, II = extension of the original storage strategy, III = creation of a new interface, IV = no change;

**Models:** R = relational, C = column, K = key/value, J = JSON, X = XML, G = graph, D = RDF, O = object, - = no support;

**Platforms:** N = Native Machine Code, J = Java/JVM, W = Win, L = Linux, U = Unix (e.g. BSD), M = macOS, S = Solaris, Z = z/OS, C = Cloud, H = Hybrid Cloud, - = no support, * = support for old versions.

## Examples of Multi-Platform Databases 2/2

<table>
<thead>
<tr>
<th>Type</th>
<th>DBMS</th>
<th>Ext.</th>
<th>Models</th>
<th>Query Languages</th>
<th>Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key/value</td>
<td>Riak KV</td>
<td>I</td>
<td>R--K---</td>
<td>Solr</td>
<td>N--LUM--CH</td>
</tr>
<tr>
<td></td>
<td>c-treeACE</td>
<td>III</td>
<td>R--K--G--</td>
<td>SQL</td>
<td>N--WLUMS--C--</td>
</tr>
<tr>
<td></td>
<td>Oracle NoSQL DB</td>
<td>III</td>
<td>R--K--GD-</td>
<td>SQL</td>
<td>-JWLUMS--C--</td>
</tr>
<tr>
<td>Document</td>
<td>Cosmos DB</td>
<td>I</td>
<td>--CKJ----</td>
<td>SQL-like</td>
<td>N----------C--</td>
</tr>
<tr>
<td></td>
<td>ArangoDB</td>
<td>II</td>
<td>--KJ--G--</td>
<td>SQL-like AQL</td>
<td>N-WL-M--C--</td>
</tr>
<tr>
<td></td>
<td>MongoDB</td>
<td>II</td>
<td>--KJ----</td>
<td>JSON-based</td>
<td>N-WL-M--C--</td>
</tr>
<tr>
<td></td>
<td>Couchbase</td>
<td>III</td>
<td>--KJ-----</td>
<td>SQL-based NQL</td>
<td>N-WL-M--CH</td>
</tr>
<tr>
<td></td>
<td>MarkLogic</td>
<td>III</td>
<td>----JX-D0</td>
<td>XPath, XQuery, SQL-like</td>
<td>N-WL-M--CH</td>
</tr>
<tr>
<td>Graph</td>
<td>OrientDB</td>
<td>II</td>
<td>--KJ--G--</td>
<td>Gremlin, extended SQL, SPARQL</td>
<td>N-WLUM--CH</td>
</tr>
<tr>
<td>Object</td>
<td>InterSystems Caché</td>
<td>III</td>
<td>R--JX--O</td>
<td>SQL with object extensions</td>
<td>N-WLUMS--CH</td>
</tr>
</tbody>
</table>

**Legend:** Ext.: I = adoption of a new storage strategy, II = extension of the original storage strategy, III = creation of a new interface, IV = no change;  
Models: R = relational, C = column, K = key/value, J = JSON, X = XML, G = graph, D = RDF, O = object, - = no support;  
Platforms: N = Native Machine Code, J = Java/JVM, W = Win, L = Linux, U = Unix (e.g. BSD), M = macOS, S = Solaris, Z = z/OS, C = Cloud, H = Hybrid Cloud, - = no support, * = support for old versions.
Multi-Platform Development of DBMS

- **Native Binaries via C/C++**
  - support of a new platform: porting code is necessary
  - code close to hardware, fast execution
  - direct access to native libraries
  - doesn't run in browser
  - most server DBMS: C/C++ code

- **Java/Java Virtual Machine (JVM)**
  - runs on many platforms (without porting code)
  - interpreted bytecode, via Just-In-Time compilation **comparable speed to native execution**
  - no direct access to native libraries
  - does neither run on iPhone nor in browser
  - many NoSQL/NewSQL/Cloud DBMS: Java (or JVM language like Scala) code

- **Code generation for query processing** via C/C++ or Janino-Compiler (JVM)
Multi-Platform Development with Kotlin

Targets:

- **Most target platforms are supported**
- **Splitting the project in platform-independent and platform-dependent code**
  - Platform-dependent code can be partly coded in the programming language of the target platform (e.g., Java for JVM, JS for Web)
- **Enables one code repository for various target platforms**
  - Sharing of code between server & (various) clients
- **Avoids efforts to port code**
  (into other programming languages)
Multi-Platform Development with Kotlin

- **Common Module**
  - Code independent of platforms containing declarations for platform dependent code without implementation, e.g.:

```
expect fun formatString(source: String, vararg args: Any): String
expect annotation class Test
```

- **Platform Module**
  - Implementation of within the common module declared platform-dependent code (and other platform-dependent code), e.g.:

```
actual fun formatString(source: String, vararg args: Any) = String.format(source, args)
actual typealias Test = org.junit.Test
```

- **Regular Module**
  - depend on platform modules or platform modules depend on this module

- **However:** High compilation times, faster: Including different sets of source code directories for different targets and configurations (e.g., centralized, Cloud, P2P, browser, ...)

---
Data sizes are growing faster than computing capacity of single CPU

Parallel/distributed computing to overcome limitations of single CPUs
Data Sizes

Size:

Binary: $2^1$ Byte $2^{10}$ Kibi $2^{20}$ Mebi $2^{30}$ Gibi $2^{40}$ Tebi $2^{50}$ Pebi $2^{60}$ Exbi $2^{70}$ Zebi $2^{80}$ Yobi

Decimal: $10^0$ Byte $10^{3}$ Kilo $10^6$ Mega $10^9$ Giga $10^{12}$ Tera $10^{15}$ Peta $10^{18}$ Exa $10^{21}$ Zetta $10^{24}$ Yotta

Data:

Office Internet Big Data*

Company:

SMEs Global Player

Devices:

IoT Device Cluster Multi-Cloud
Embedded Mobile Cloud
Historical Home Computer Server Desktop

Databases:

Main Memory Hardware Cloud
Centralized IoT

Platforms:

Desktop Cloud
Web/Mobile Fog/Edge/Dew

SMEs: Small and medium-sized enterprises  * social media, search engines

Atoms on Earth

$2^{167}$ $2^{80}$
Amdahl's versus Gustafson's law

- **Amdahl's law**
  - a sequential part of the overall algorithm limits overall speedup (in the context of fixed problem/data size)

  ![Graph showing speedup vs number of nodes for Amdahl's law](image)

- **Gustafson's law**:
  - programmers tend to set the size of problems to fully exploit the computing power that becomes available as the resources improve
  - if faster equipment or more nodes are available, larger problems can be solved within the same time
PACELC Theorem as Refinement of CAP

- In case of network partitions (P):
  - Guarantee of either Availability (A) or Consistency (C) (like CAP theorem)
- In normal operation without network partitions errors: "Else (E)"
  - Guarantee of either small Latency (L) or strong Consistency (C)
- NoSQL-DBMS
  - some with several configuration possibilities
  - challenge for hybrid: transparent global approaches supporting different PACELC properties for different partitions at the same time

<table>
<thead>
<tr>
<th>Distributed DBMS</th>
<th>P+A</th>
<th>P+C</th>
<th>E+L</th>
<th>E+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>DynamoDB, Cassandra, Cosmos DB, Riak</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Couchbase, FaunaDB</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>VoltDB/H-Store, Megastore, BigTable/HBase, MySQL Cluster</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>MongoDB</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>PNUTS</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Hazelcast IMDG</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Platform-specific types of DBMS

- **Parallel**
  - Server
  - Main Memory
  - Cluster
  - Hardware-Accelerated

- **Distributed**
  - Federated
  - Cloud
  - Web Cloud
  - DBMS on Small Devices

- **Processing in Cloud**
  - Mobile
  - IoT
  - Embedded

- **Other**
  - Smart SSD
  - Many-Core
  - CPU
  - FPGA
  - Quantum Computer
  - Quantum Annealer
  - Mobile
  - IoT
  - Embedded
  - Processing in Cloud
Cloud Computing Architecture

- **Large cluster** with up to several thousand nodes
- **Replication** of data blocks (default 3 times)
- **Simple error** detection and recovery by job repetition
Capacity-Cost Performance

- High capital expenditures for buying new hardware
- Additional costs
- Area of loosing clients!

(CPU, RAM, HDD, SDD, …)

Capacity

Time

Predicted capacity

Traditional hardware

Currently needed capacity

Automatic Cloud capacity
Cloud DBMS

Scalability
- Petabytes of data
- Thousands of Computers

Flexibility
- Processing of any data format
- Schemaless/without schema

Traditional DBMS

High performance
- Only for read-heavy workloads
- Updates are relatively slow

Uniform Data format
- Separation of schema and content
Cloud DBMS

- (Relatively) cheap (commodity-) hardware
- Efficient and simple fault-tolerant mechanisms
  - Dealing with frequent errors (hardware/communication)

Traditional DBMS

- Few high-end server
  - Few hardware crashes
- Transactions: Guaranty of Atomicity, Consistency, Isolation, Durability properties
  - Assumption: Error case is seldom
P-LUPOSDATE - Stack

LUPOSDATE

HBase

HDFS

Pig

MapReduce
P-LUPOSDATE - Bloomfilter and Query Processing
# P-LUPOSDATE - Indexing Scheme

## Table: S_PO

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rowkey 1</td>
<td><code>&lt;http://domain.org/Reifen&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>Column Family 1: Hexa</code></td>
</tr>
</tbody>
</table>

## Table: O_SP

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rowkey 2</td>
<td><code>&lt;http://domain.org/Motor&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>Column Family 1: Hexa</code></td>
</tr>
</tbody>
</table>

## Table: P_SO

## Table: SP_O

## Table: SO_P

## Table: PO_S
P-LUPOSDATE - Experimental Evaluation

1 Billion Triples

Speedup between 1 and 2.24 (Avg.: 1.34)

with Bloomfilter
without Bloomfilter
Typical **Big Data Analytics Stack**
(e.g. Spark, Flink, Storm)

**Streaming Processing**
- **Spark**: Streaming
- **Flink**: DataStream

**Scripting Language/Applications**
- **Spark**: Spark SQL
- **Flink**: Table API & SQL

**Machine Learning**
- **Spark**: MLib
- **Flink**: FlinkML

**Graph Processing**
- **Spark**: GraphX
- **Flink**: Gelly

---

**Execution Engine**

**Resource Management**

**Database**

**Storage**

**Streams**
Evolution of Big Data Analytics Engines

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Model</td>
<td>MapReduce</td>
<td>DAG Dataflows</td>
<td>Resilient Distributed Datasets (RDD)</td>
<td>Cyclic Dataflows</td>
<td>?</td>
</tr>
<tr>
<td>Engine</td>
<td>Hadoop</td>
<td>TEZ</td>
<td>Spark</td>
<td>Flink</td>
<td>?</td>
</tr>
</tbody>
</table>

1. Generation: Batch Processing
2. Generation: Batch + Interactive Processing
3. Generation: Batch + Near-Real-Time + Iterative Processing
4. Generation: Batch + Real-Time Streaming + Native Iterative Processing
5. Generation: ?
What is missing: Maximizing local Joins

Iterative Optimization
Migrate node: -3+1=-2 edges ⇒ more local joins
Platform-specific types of DBMS

- Server
- Main Memory
- Cluster
- Hardware-Accelerated
- Smart SSD
- Many-Core
- CPU
- FPGA
- Quantum Computer
- Quantum Annealer

- Federated
- Cloud
- Web Cloud
- DBMS on Small Devices
- Mobile
- IoT
- Embedded

- Processing in Cloud
  - Fog
  - Edge
  - Dew
Architectures of Emergent Hardware

- **Multi-Core CPU**
  - Cores: ~10
  - Core Complexity: Complex (optimized for single thread performance)
  - Computational Model: MIMD + SIMD
  - Parallelism: Thread and Data Parallel
  - Memory Model: Shared
  - Power: 150 W
  - Database Op.: Query Optimization (Enumeration of Plans), Concurrency Control

- **Many-Core CPU**
  - Cores: ~100
  - Core Complexity: Simple
  - Computational Model: SIMD
  - Parallelism: Data Parallel
  - Memory Model: Distributed
  - Power: 200 W
  - Database Op.: Efficient Processing Of Query Processing

- **Graphics Processing Unit (GPU)**
  - Cores: ~1000
  - Core Complexity: Simple
  - Computational Model: SIMD
  - Parallelism: Data Parallel
  - Memory Model: Distributed
  - Power: 250 W

- **Field Programmable Gate Arrays (FPGA)**
  - Cores: ~100 000
  - Core Complexity: Simple
  - Computational Model: Data-Flow
  - Parallelism: Arbitrary
  - Memory Model: Distributed
  - Power: 50 W

- **Quantum Computer/Annealer**
  - Cores: ~50 / ~2000 qubits
  - Core Complexity: Complex
  - Computational Model: Universal/Adiabatic Quantum computing
  - Parallelism: Quantum Logic Gates/Fluctuation (glob. Min.)
  - Memory Model: Quantum Superposition
  - Power: 25 KW

Legend:
- Purple: Computational Unit
- Green: Execution Controller
- Orange: Interconnection Network
- Magenta: On-Chip Memory
General Purpose Graphics Processing Unit (GPGPU)

- turns the massive computational power of a modern graphics accelerator's shader pipeline into general-purpose computing power
- Single instruction, multiple data (SIMD)
- Up to several thousand computing cores
- Programming languages for SIMD computations
  - Open Computing Language (OpenCL): Vendor-independent programming standard
  - CUDA (formerly Compute Unified Device Architecture): NVIDIA-dependent parallel computing platform and API model
  - Open Graphics Library (OpenGL): mainly cross-language, cross-platform API for rendering 2D and 3D vector graphics
Approximate Search in Adaptive Radix Tree (ART) on GPGPUs

- **Levenshtein-distance**: number of operations to transform one string into another:

<table>
<thead>
<tr>
<th>institution</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>o</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>m</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>i</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>n</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>g</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

- **Speedup over 4** dependent on ART properties (1.43 for real-world BTC data)

**Legend**
- **CPU**: institute
- **GPU**: institute, ed: 4
- **Node**: search term with edit distance
- **type prefix**: parallel searches
- **inner Node**: edit distance reset buffer
- **reset to next saved edit distance**: calculation of edit distance for character in previous column is unchanged, this is static memory and will be overwritten
- **leaf**: initial status of the dynamic column
- **edge in the tree**: sequential executions
- **key value**: parallel execution in the tree
- **segment execution with key x**: reset to next saved edit distance

Multi-model Multi-platform (HM3P) Databases
Emergent Hardware for Databases with focus on Semantic Web: Semantic Hybrid
real-world BTC data)
High-End Parallel GPU System: DGX-2

- **16X NVIDIA Tesla V100 32GB**
  - Total: GPU Memory: 512GB total
  - Performance: 2 petaFLOPS
  - CUDA® Cores: 81,920
  - Tensor Cores: 10,240

- **12X NVSwitches**
  - 2.4 TB/sec bi-section bandwidth

- **2X GPU Boards**
  - Per Board: 8 V100 32GB GPUs
  - 6 NVSwitches interconnected by Plane Card

- **Dual 10/25 GigE**

- **Eight EDR Infiniband/100 GigE**
  - 1,600 Gb /sec Total
  - Bi-directional Bandwidth

- **PCIe Switch Complex**

- **2X Intel Xeon Platinum CPUs**
  - 8168, 2.7 GHz (Turbo 3.7GHz), 24-cores (48 threads)

- **30 TB NVME SSDs**
  - Internal Storage

- **1.5 TB System Memory**
  - Image by NVIDIA
High-End Parallel GPU System: **DGX A100**

8X **NVIDIA A100**
- Total: 320GB total
- Performance:
  - 5 petaFLOPS (AI)
  - CUDA® Cores: 55,296
  - Tensor Cores: 3,456
- 12 NVLinks/GPU
- 600 GB/s bandwidth between GPUs

9 Mellanox ConnectX-6 VPI HDR InfiniBand/200 GB Ethernet: 2,025 Gb/s Total bi-dir. bandwidth

2X **AMD Rome CPUs Epyc 7742**
- 2.25 GHz
- Turbo 3.4GHz
- 64-cores
- 128 threads
- 1 TB System Memory

6X **NVSwitches**
- 4.8 TB/s bi-section bandwidth

15 TB **GEN4 NVME SSDs**
- Internal Storage
- 25.6 GB/s max. bandwidth

Images by NVIDIA
Field-Programmable Gate Array (FPGA)

- contains an array of programmable logic blocks and a hierarchy of reconfigurable interconnects
- Specification of configuration typically by hardware description language (HDL)
- Recently High Level Synthesis (e.g., OpenCL) more mature (but still performance-critical parts should not be implemented in OpenCL)
- Long synthesis time
LUPOSDATE on FPGA – Query Processing

- Generation of Reconfigurable Modules (RMs) at system deployment time
- Selection of RMs and configuration into Reconfigurable Partitions at system runtime $\leadsto$ avoids long synthesis time
Configuring the **Semi-Static Operator Graph**

RP: Reconfigurable Partition
SRE: Semi-static Routing Element

SP²B

Query 4

---

S. Werner, Hybrid Architecture for Hardware-accelerated Query Proc. in Semantic Web DB based on Runtime Reconfigurable FPGAs, 2017
LUPOSDATE on FPGA – Benchmark Results

- **Reconfiguration** reduced from about half hour to few milliseconds (< 20 ms for all queries) when using semi-static operator graphs

- **SP^2B Benchmark**
  - Dataset sizes from 66 to 262 million triples
  - Speedups between 4 and 32 times
    (dependent on query and dataset size)
Hybrid Index - FPGA Accelerated Index

- $\text{B}^+\text{-Tree (compact static index: CSB}^+\text{-Tree)}$: Speedup of 2.3

  Larger speedups possible via pipelining and usage of memory hierarchies (currently only BRAM)
Quantum Computer

- use of quantum-mechanical phenomena such as superposition and entanglement to perform computation
- Different types of quantum computer, e.g.
  - Digital Quantum Computer
    - uses quantum logic gates to do computation
    - measurement (sometimes called observation) assigns the observed variable to a single value
  - Quantum Annealing
    - metaheuristic for finding the global minimum of a given objective function over a given set of candidate solutions
    - i.e., some way to solve a special type of mathematical optimization problem
Using **Hardware Accelerator** for optimizing Transaction Schedules
2 Phase Locking (2PL) versus Strict Conservative 2PL

• **required locks** to be determined by
  - static analysis of transaction, or if static analysis is not possible:
    - an additional phase at runtime before transaction processing
Optimizing Transaction Schedules

- Job shop schedule problem (JSSP):
  - Multi-Core CPU
    - Process whole job (here transaction) on core X
  - Schedule: ∀ cores: Sequence of jobs to be processed
  - What is the optimal schedule for minimal overall processing time?

- Additionally to JSSP:
  Blocking transactions not to be processed in parallel

- Example:

Black: Blocking transactions

- JSSP is among the hardest combinatorial optimizing problems

- Hardware accelerating the optimization of transaction schedules

---

Optimizing Transaction Schedules via Quantum Annealing

- Scenario: **Strict conservative 2-Phase Locking**
  - Preclaiming of all locks at *Begin of Transaction* (avoids deadlocks)
  - Holding all locks until *End of Transaction* (avoids cascading aborts)

- Solution formulated as set of binary variables
  - $X_{i,j,s}$ is 1 iff transaction $t_i$ is started at time $s$ on machine $m_j$, otherwise 0

- Example:

  ![Diagram showing transaction schedule and solution](image)

  - Solution: $X_{1,1,0}$, $X_{3,1,2}$, $X_{4,2,0}$, $X_{7,2,1}$, $X_{6,2,3}$, $X_{5,2,6}$, $X_{2,3,0}$, $X_{8,3,5}$
Optimizing Transaction Schedules via Quantum Annealing

- **Transaction Model**
  - \( T \): set of transactions with \(|T| = n\)
  - \( M \): set of machines with \(|M| = k\)
  - \( O \subseteq T \times T \): set of blocking transactions
  - \( l_i \): length of transaction \( i \)
  - \( R \): maximum execution time
  - upper bound \( r_i = R - l_i \) for start time of transaction \( i \)

- **Example**
  - \( T = \{t_1, t_2, t_3\} \), \( n=3 \)
  - \( M = \{m_1, m_2\} \), \( k=2 \)
  - \( O = \{(t_2, t_3)\} \)
  - \( l_1 = 2, l_2 = 1, l_3 = 1 \)
  - \( R = 2 \)
  - \( r_1 = 0, r_2 = 1, r_3 = 1 \)

- **Quadratic unconstrained binary optimization (QUBO) problems** (solving is NP-hard)
  - A QUBO-problem is defined by \( N \) weighted binary variables \( X_1, \ldots, X_N \in \{0, 1\} \), either as linear or quadratic term to be minimized:
  \[
  \sum_{0<i\leq N} w_i X_i + \sum_{i\leq j\leq N} w_{ij} X_i X_j, \text{ where } w_i, w_{ij} \in \mathbb{R}
  \]
Optimizing Transaction Schedules via Quantum Annealing

- **Valid Solution**
  - A: each transaction starts exactly once
    \[ A = \sum_{i=1}^{n} \left( \sum_{j=1}^{k} \sum_{s=0}^{r_i} X_{i,j,s} - 1 \right)^2 \]
    - transactions
    - machines
    - start times
  
  - B: transactions cannot be executed at the same time on the same machine
    \[ B = \sum_{j=1}^{k} \sum_{i_1=1}^{n-1} \sum_{s_1=0}^{r_i} \sum_{i_2=i_1+1}^{n} \sum_{s_2=q}^{p} X_{i_1,j,s_1} X_{i_2,j,s_2} \text{ for } q = \max\{0, s_1 - l_{i_2} + 1\}, p = \min\{s_1 + l_{i_1}, r_{i_2}\} \]
    - transactions without \( t_n \) remaining transactions
    - machines
    - start times
    - invalid start times

  - C: transactions that block each other cannot be executed at the same time
    \[ C' = \sum_{\{t_{i_1}, t_{i_2}\} \in O} \sum_{j_1=1}^{k} \sum_{s_1=0}^{r_i} \sum_{j_2 \in J} \sum_{s_2=q}^{p} X_{i_1,j_1,s_1} X_{i_2,j_2,s_2} \text{ for } J = \{1, \ldots, k\} \setminus \{j_1\}, q = \max\{0, s_1 - l_{i_2} + 1\}, p = \min\{s_1 + l_{i_1}, r_{i_2}\} \]
    - blocking transactions
    - start times
    - invalid start times
Optimizing Transaction Schedules via Quantum Annealing

- **Optimal Solution**
  - D: minimizing the maximum execution time
  
  \[ D = \sum_{i=1}^{n} \sum_{j=1}^{k} \sum_{s=0}^{r_i} w_{s+l_i} X_{i,j,s}, \text{ where } w_{s+l_i} = \frac{(k + 1)^{s+l_i-1}}{(k + 1)^R} < 1 \]

  - Increasing weights: Weight of step n is larger than of all preceding steps 1 to n-1 \( \Rightarrow \) preferring transactions ending earlier
  - Weights in A, B and C \( \geq 1 \)
    \( \Rightarrow \) first priority is validity, second priority is optimality

- **Overall Solution**
  - Minimize \( P = A + B + C + D \)
Optimizing Transaction Schedules via Quantum Annealing

- Experiments on real Quantum Annealer (D-Wave 2000Q cloud service)
  - first minute free
  (afterwards too much for our budget)
- Versus Simulated Annealing on CPU
- Preprocessing time/Number of QuBits: $O((n \cdot k \cdot R)^2)$

<table>
<thead>
<tr>
<th>Fig.</th>
<th>$k$</th>
<th>$n$</th>
<th>$R$</th>
<th>$O$</th>
<th>$l_1, \ldots, l_n$</th>
<th>$r_1, \ldots, r_n$</th>
<th>req. var.</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>2</td>
<td>8</td>
<td>2</td>
<td>{}</td>
<td>8, 4</td>
<td>0, 4</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>$(t_1, t_3)$</td>
<td>4, 5, 1</td>
<td>1, 0, 4</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>$(t_2, t_4)$</td>
<td>3, 2, 1, 2</td>
<td>1, 2, 3, 2</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>$(t_1, t_2), (t_4, t_5)$</td>
<td>1, 1, 1, 1, 1</td>
<td>1, 1, 1, 1, 1</td>
<td>10</td>
</tr>
</tbody>
</table>
Platform-specific types of DBMS
IoT Architectures

Cloud Computing

Bottleneck

Fog Computing

Edge Computing

Dew Computing
Platform-specific types of DBMS

- Parallel
  - Server
  - Main Memory
  - Cluster
  - Hardware-Accelerated
- Distributed
  - Federated
  - Cloud
  - Web Cloud
  - DBMS on Small Devices
- Mobile
  - IoT
  - Embedded
- Processing in Cloud
  - Fog
  - Edge
  - Dew
Mobile DBMS integrated into Architecture for Mobile Phones
Platform-specific types of DBMS
## Features of different types of databases

<table>
<thead>
<tr>
<th>Feature</th>
<th>DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Main Memory</td>
</tr>
<tr>
<td>Scalability</td>
<td></td>
</tr>
<tr>
<td>Transaction rates</td>
<td>+</td>
</tr>
<tr>
<td>Intra-Transaction Parallelism</td>
<td>+</td>
</tr>
<tr>
<td>Atomicity</td>
<td>+</td>
</tr>
<tr>
<td>Durability</td>
<td></td>
</tr>
<tr>
<td>Consistency</td>
<td>+</td>
</tr>
<tr>
<td>Extensibility</td>
<td></td>
</tr>
<tr>
<td>Schemaless</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td></td>
</tr>
<tr>
<td>Transparency of Distribution</td>
<td></td>
</tr>
<tr>
<td>Geographical Distribution</td>
<td></td>
</tr>
<tr>
<td>Mobility</td>
<td></td>
</tr>
<tr>
<td>Node Autonomy</td>
<td></td>
</tr>
<tr>
<td>Heterogeneity of DBMS</td>
<td></td>
</tr>
<tr>
<td>Administration</td>
<td>o</td>
</tr>
<tr>
<td>Hardware Costs</td>
<td></td>
</tr>
<tr>
<td>Reasoning</td>
<td></td>
</tr>
</tbody>
</table>
Hybrid Multi-Model Multi-Platform (HM3P) Database

- How to integrate the features of different types of databases into one single database running also on different platforms?
Challenges for HM3P Databases 1/2

- developing only one code base for the different platforms, but not introducing performance overhead in comparison to single platform databases
- identifying common properties of several platforms and reusing those approaches (like fault tolerance mechanisms) in different combinations, which are best suitable for these considered platforms
- data distribution among different platforms (applying different data distribution approaches as well)
- data distribution strategies considering overall the different properties of used platforms and models (like fast reads on parallel servers (using relational databases) and fast updates in cloud databases)
Challenges for HM3P Databases 2/2

- **query optimization and other database tasks across different platforms**, which apply different database approaches
- dealing with and **integrating different privacy and security mechanisms** supporting different privacy and security levels in the different platforms (with research e.g. on querying heterogeneous encrypted data)
- **concurrency control** approaches of different type have to be combined and work in cooperation (like 2 phase locking for server platforms and optimistic concurrency control for P2P networks)
- **combining different types of databases** (on different platforms) to offer the best of these databases and platforms under one hood to applications and users transparently or via intelligent integration into query language and API, e.g.,
  - guaranteeing atomicity and isolation in transactions for the data stored on a parallel server, but not for those data in the cloud supporting fast updates
Semantic Hybrid Multi-Model Multi-Platform (SHM3P) Database

Single instance of **SHM3P Database** offers (fully cross-platform optimized) functionality of & replaces

- IoT DB *On the Edge*
- Main-Memory DB *GPU-accelerated Parallel Server*
- Quantum DB *Quantum Computer*
- Cloud DB *Cloud&Infrastructure*

**Reasoning:**
- Lightweight reasoning on large data sizes of IoT devices
- Heavyweight reasoning on moderate data sizes
- Heavyweight reasoning on large data sizes
- Reasoning on small data sizes of mobile devices

---

**How to integrate the different reasoning capabilities and requirements into one transparent global reasoner?**

- **How to integrate the semantic layer between different types of databases and support semantic processing specialities like reasoning over the boundaries of different platforms?**
Challenges for SHM3P Databases

- integrating different data models in a semantic layer on top of the underlying data models
- efficient transformations from and to the semantic model in an operational system
- developing efficient semantic querying and reasoning over the integrated data of different models
- global reasoning over reasoners running on different platforms supporting some kind of distributed heterogeneous reasoning
- developing a combination of stream reasoning over streaming data (e.g. of IoT devices) with static reasoning over large-scale data sets (stored e.g. in clouds)
- supporting transactions over semantic data by integrating the reasoner in transaction synchronization
Proposals for Cooperation & Collaborations

- Contributions to luposdate3000 are welcome: https://github.com/luposdate3000/luposdate3000
  - current status: SMP DBMS, soon SHMP DBMS
- Any other computer science topic in my expertise area
- Please contact me: groppe@ifis.uni-luebeck.de
Summary and Conclusions

- Different **data models** and their special features
  - Multi-Model Databases

- Different **platforms and a need for different types of databases**
  - Different features
  - Multi-Platform Databases

- Databases spanning over different platforms in operation (**supporting multiple data models**)
  - Hybrid Multi-Model Multi-Platform (HM3P) Databases
CfPs: My workshops at best-ranked DB conferences

International Workshop on

Big Data in Emergent Distributed Environments (BiDEDE 2021)
in conjunction with the 2021 ACM SIGMOD Conference (online)

- [https://www.ifis.uni-luebeck.de/~groppe/bidede/](https://www.ifis.uni-luebeck.de/~groppe/bidede/)
- Submission: March 18, 2021
- Workshop (online): June 20, 2021

International Workshop on

Very Large Internet of Things (VLIoT 2021)
in conjunction with the 2021 VLDB Conference in Copenhagen, Denmark

- [https://www.ifis.uni-luebeck.de/~groppe/vliot/](https://www.ifis.uni-luebeck.de/~groppe/vliot/)
- Submission: April 5, 2021
- Workshop (hybrid): August 16, 2021