Lecture

Quantum Computing

(CS5070)

Quantum Annealing Versus Grover's Search: Optimizing Transaction Schedules

Professor Dr. rer. nat. habil. Sven Groppe

https://www.ifis.uni-luebeck.de/index.php?id=groppe
Platform-specific types of DBMS
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  - ...this concept is no longer applicable to the database market...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

**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?

- **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**

- Multi-Model
- One Model

**Platform Diversity**

- Single Platform
- Multiple Platforms
- Hybrid Multiple Platforms

**Legend:**
- S: Semantic
- MP: Multi-Platform
- MM: Multi-Model
- M3P: MM MP
- H: Hybrid

- **Support of semantic layer**

- **State-of-the-art partly/rudimentarily addressed**

- **Visionary/single attempts (e.g. hybrid cloud)**

- **Global semantic layer without semantic layer**
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:**
- JVM
  - Desktop
  - Server
  - Android
- JS
  - Browser
  - Server NodeJS
- LLVM/Native
  - Win (X64 / X86)
  - Linux (arm64/32 / 32Hfp / Mips(elf)32 / X64)
  - MacOS (X64)
  - iOS (arm32/64 / X64)
  - watchOS (arm32/64 / X86)
  - Android (arm32/64)
  - WebAssembly (wasm32)

- 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.:
    ```kotlin
    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.:
    ```kotlin
    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, ...)

Quantum Computing
Quantum Annealing Versus Grover’s Search: Optimizing Transaction Schedules

Institut für Informationssysteme | Prof. Dr. habil. S. Groppe
The Power of Multi-Platform: LUPOSDATE3000

- ultra-fast in jvm...

...but also enabling web demos running completely in the browser!

B. Warnke, M.W. Rehan, S. Fischer, S. Groppe: Flexible data partitioning schemes for parallel merge joins in semantic web queries in: BTW'21

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

- Variant of 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:

```
Transaction schedule
```

- JSSP is among the hardest combinatorial optimizing problems
- \( \text{⇒} \) Hardware accelerating the optimization of transaction schedules
## Architectures of Emergent Hardware

<table>
<thead>
<tr>
<th>Multi-Core CPU</th>
<th>Many-Core CPU</th>
<th>Graphics Processing Unit (GPU)</th>
<th>Field Programmable Gate Arrays (FPGA)</th>
<th>Quantum Computer/Annealer</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Multi-Core CPU" /></td>
<td><img src="image2.png" alt="Many-Core CPU" /></td>
<td><img src="image3.png" alt="Graphics Processing Unit (GPU)" /></td>
<td><img src="image4.png" alt="Field Programmable Gate Arrays (FPGA)" /></td>
<td><img src="image5.png" alt="Quantum Computer/Annealer" /></td>
</tr>
<tr>
<td><strong>Cores</strong></td>
<td>~10</td>
<td>~100</td>
<td>~1000</td>
<td>~100/~5000 qubits</td>
</tr>
<tr>
<td><strong>Core Complexity</strong></td>
<td>Complex (optimized for single thread performance)</td>
<td>Simple</td>
<td>Complex</td>
<td></td>
</tr>
<tr>
<td><strong>Computational Model</strong></td>
<td>MIMD + SIMD</td>
<td>SIMD</td>
<td>Data-Flow</td>
<td>Universal/Adiabatic Quantum computing</td>
</tr>
<tr>
<td><strong>Parallelism</strong></td>
<td>Thread and Data Parallel</td>
<td>Data Parallel</td>
<td>Arbitrary</td>
<td>Quantum Logic Gates/Fluctuation (glob. Min.)</td>
</tr>
<tr>
<td><strong>Memory Model</strong></td>
<td>Shared</td>
<td>Distributed</td>
<td>Quantum Superposition</td>
<td></td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>150 W</td>
<td>200 W</td>
<td>250 W</td>
<td>50 W</td>
</tr>
<tr>
<td><strong>Database Op.</strong></td>
<td>Query Optimization (Enumeration of Plans), Concurrency Control</td>
<td>Efficient Processing Of</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Legend:**
- Purple: Computational Unit
- Green: Execution Controller
- Yellow: Interconnection Network
- Pink: On-Chip Memory

Quantum Computer

- use of quantum-mechanical phenomena such as superposition and entanglement to perform computation
- Different types of quantum computer, e.g.
  - Universal Quantum Computer
    - uses quantum logic gates arranged in a circuit 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
Quantum versus Simulated Annealing

- **Energy/Cost Function**
  - Local Minima
  - Quantum Tunnel Effect
  - Classical Path for Simulated Annealing
  - Quantum Tunnel Effect
  - Global Minimum

**System Configuration/Solution**
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

- Multi-Core CPU
  - Process whole transaction on core X
- 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:

  - 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

- **Valid Solution**
  - A: each transaction starts exactly once

\[
A = \sum_{i=1}^{n} \left( \sum_{j=1}^{k} \sum_{s=0}^{r_i} X_{i,j,s} - 1 \right)^2
\]

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

\[
A = (X_{1,1,0} + X_{1,2,0} - 1)^2 + (X_{2,1,0} + X_{2,1,1} + X_{2,2,0} + X_{2,2,1} - 1)^2 + (X_{3,1,0} + X_{3,1,1} + X_{3,2,0} + X_{3,2,1} - 1)^2
\]
Optimizing Transaction Schedules via QA

- **Valid Solution**
  - $B$: transactions cannot be executed at the same time on the same machine

$$B = \sum_{j=1}^{k} \sum_{i_1=1}^{n-1} r_{i_1} \sum_{s_1=0}^{n-1} \sum_{i_2=i_1+1}^{p} X_{i_1, j, s_1} X_{i_2, j, s_2} \text{ for } q = \max\{0, s_1 - l_{i_2} + 1\}, p = \min\{s_1 + l_{i_1}, r_{i_2}\}$$

### Example:

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

$$B = X_{1,1,0}X_{2,1,0} + X_{1,1,0}X_{2,1,1} + X_{1,1,0}X_{3,1,0} + X_{1,1,0}X_{3,1,1} + X_{2,1,0}X_{3,1,0} + X_{2,1,1}X_{3,1,1} + X_{1,2,0}X_{2,2,0} + X_{1,2,0}X_{2,2,1} + X_{1,2,0}X_{3,2,0} + X_{1,2,0}X_{3,2,1} + X_{1,2,1}X_{3,2,1} + X_{2,2,0}X_{3,2,0} + X_{2,2,1}X_{3,2,1}$$
**Optimizing Transaction Schedules via Quantum Annealing**

- **Valid Solution**
  - C: transactions that block each other cannot be executed at the same time

  \[
  C = \sum_{\{t_{i_1}, t_{i_2}\} \in O} \sum_{j_1=1}^{k} \sum_{s_1=0}^{r_{i_1}} \sum_{j_2 \in J} \sum_{s_2=q}^{p} X_{i_1, j_1, s_1} X_{i_2, j_2, s_2} \quad \text{for } J = \{1, \ldots, k\} \setminus \{j_1\}, q = \max\{0, s_1 - l_{i_2} + 1\}, p = \min\{s_1 + l_{i_1}, r_{i_2}\}
  \]

**Example:**

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

\[
C = X_{2,1,0}X_{3,2,0} + X_{2,1,1}X_{3,2,1} + X_{2,2,0}X_{3,1,0} + X_{2,2,1}X_{3,1,1}
\]
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

**Example:**
- \(R = 2\)
- \(T = \{t_1, t_2, t_3\} \text{ with } |T| = n = 3\)
- \(M = \{m_1, m_2\} \text{ with } |M| = k = 2\)
- \(O = \{(t_2, t_3)\}\)
- \(l_1 = 2, l_2 = 1, l_3 = 1\)
- \(r_1 = 0, r_2 = 1, r_3 = 1\)
- \(D = \frac{3}{9} X_{1,1,0} + \frac{3}{9} X_{1,2,0} + \frac{1}{9} X_{2,1,0} + \frac{3}{9} X_{2,1,1} + \frac{1}{9} X_{2,2,0} + \frac{3}{9} X_{2,2,1} + \frac{1}{9} X_{3,1,0} + \frac{3}{9} X_{3,1,1} + \frac{1}{9} X_{3,2,0} + \frac{3}{9} X_{3,2,1}\)
Optimizing Transaction Schedules via Quantum Annealing

• **Overall Solution**
  - Minimize $P = A + B + C + D$

Optimal schedules (transaction 1 in blue, transaction 2 in green and transaction 3 in red) for our example:

- Machine 1: $X_{1,1,0}, X_{2,2,0}, X_{3,2,1}$
- Machine 2: $X_{1,1,0}, X_{2,2,1}, X_{3,2,0}$
- Machine 1: $X_{1,2,0}, X_{2,1,0}, X_{3,1,1}$
- Machine 2: $X_{1,2,0}, X_{2,1,1}, X_{3,1,0}$

The result of $P$ is the following value for all 4 different (optimal) schedules:

$P = A + B + C + D = -3 + 0 + 0 + \frac{7}{9} = -\frac{2}{9}$

If the offset is added (optional), then the result is:

$P = A + B + C + D = -3 + 0 + 0 + \frac{7}{9} + 3 = \frac{7}{9}$
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>8</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>5</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>2</td>
<td>$(t_1, t_2), (t_4, t_5)$</td>
<td>1, 1, 1, 1</td>
<td>1, 1, 1, 1</td>
<td>10</td>
</tr>
</tbody>
</table>
Pipelining for further Speedup
Caching of & Reusing generated formulas to minimize preprocessing time

- Following parameters are fixed:
  - the number $k$ of machines (system does not change during runtime)
  - the number $n$ of transactions (for batches of the same size)
- We observe:
  - The (maximal) execution time $R$ and hence upper bounds of start times $r_i, \ldots, r_n$ depend on the lengths of the transactions,
  - the formulas $A, B$ and $D$ depend on the fixed parameters $k$ and $n$, and on the lengths of the transactions, and
  - the formula $C$ is a sum of sub-formulas depending on $k$ and the lengths of blocking transactions as well as the identifiers of blocking transactions.

$A \Rightarrow Caching \ A, B$ and $D$, and sub-formulas of $C$ with the key of the lengths of the transactions (orderd by lengths) (Example: using $(1, 1, 2)$ as key)

Further information in paper!
Optimizing Transaction Schedules via Quantum Computing

- Quantum Computing
- Quantum Annealing Versus Grover's Search: Optimizing Transaction Schedules

Institut für Informationssysteme | Prof. Dr. habil. S. Groppe
Grover's Search Algorithm

• **Black box** function $f: \{0, \ldots, 2^b - 1\} \mapsto \{\text{true}, \text{false}\}$

• Grover's search algorithm finds one $x \in \{0, \ldots, 2^b - 1\}$, such that $f(x) = \text{true}$
  
  - if there is only one solution: $\frac{\pi}{4} \cdot \sqrt{2^b}$ basic steps each of which calls $f$

  Let $f'(b)$ be runtime complexity of $f$ for testing $x$ to be true:

  $\Rightarrow O(\sqrt{2^b} \cdot f'(b))$

  - if there are $k$ possible solutions: $O(\sqrt{\frac{2^b}{k}} \cdot f'(b))$
Overview of Optimizing Transaction Schedules via Quantum Computing

Quantum Annealing Versus Grover’s Search: Optimizing Transaction Schedules

Institut für Informationssysteme | Prof. Dr. habil. S. Groppe

Transactions

Lengths

Check:
\{0, \ldots, 2^b - 1\} \rightarrow \{True, False\}

Grover’s Search

Encoding Scheme of Solution:
\{0, \ldots, 2^b - 1\} \rightarrow \alpha

Transaction Schedule \(\alpha\)

<table>
<thead>
<tr>
<th>Core 0</th>
<th>(T_{\alpha(0,0)})</th>
<th>(T_{\alpha(0,m_1 - 1)})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core (m-1)</td>
<td>(T_{\alpha(m-1,0)})</td>
<td>(T_{\alpha(m-1,m_m - 1)})</td>
</tr>
</tbody>
</table>

Estimator for number of solutions

Code Generation

Transactions

Conflicts

\(T_0\) \(T_1\) \(T_{n-1}\)

\(T_0\) \(X\)

\(T_1\) \(X\) \(X\)

\(T_{n-1}\) \(X\)
Encoding Scheme of Transaction Schedules

```
Algo determineSchedule
Input:    p:{0, ..., 2^b-1}
Output:   \{0, ..., n-1\}^{m-1} \times 
          \{0, ..., n-1\}^{n-1}
for(x in 1..m-1)
    \mu_x = p \mod n
    p = p \div n
a = [0, ..., n-1]
for(i in 0..n-1)
    j = p \mod (n-i)
    p = p \div (n-i)
    \pi[i] = a[j]
    a[j] = a[n-i-1]
return (\mu_1, ..., \mu_{m-1}, \pi)
```

Example (n=4, m=2):

```
x = 1:
    \mu_1 = 1
    p = 7
    a = [0, 1, 2, 3]
    i = 0:  i = 1:  i = 2:  i = 3:
    j = 3  j = 1  j = 0  j = 0
    p = 1  p = 0  p = 0  p = 0
    \pi[0] = 3  \pi[1] = 1  \pi[2] = 0  \pi[3] = 2
return (1,[3,1,0,2])
```

- 29 = 000111 01 \text{ binary} \equiv \text{Core 0 } [3, 1] \mu_1 = 1 [0, 2] \text{ Core 1, some bits for permutation and some for separators}
- b = (m - 1) \cdot [\log_2(n - 1)] + [\log_2(n! - 1)]
Generated **Black Box Function**

- **Quantum computation:** circuit of quantum logic gates
  \[ \Rightarrow \text{circuit is generated} \] dependent on the concrete problem instance

- **Sketch of algo:**
  1. Determine Separators and Permutation \( O(m + n) \)
  2. Check Validity of Separators \( O(m) \)
  3. \( \forall i: \) Determine lengths of \( i \)-th transaction in permutation \( O(n \cdot \log_2 (n)) \) with decision tree over transaction number
  4. Check: Which separator configuration? For current case:
     4a. determine total runtime of core and check if it's below given limit \( O(n) \)
     4a. determine start and end times of conflicting transactions \( O(n \cdot \log_2 (\min(n, c)) + c) \) with decision tree over conflicting transactions (for \( n >> c \)) or transaction numbers (for \( c >> n \))
  5. Check: Do conflicting transactions overlap? \( O(c) \)
     \[ \sum : O(n \cdot \log_2 (n) + c) \]
## Complexity Analysis

<table>
<thead>
<tr>
<th>Approach</th>
<th>CPU</th>
<th>Quantum Computer</th>
<th>Quantum Annealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessing</td>
<td>$O(1)$</td>
<td>$O(n^2 \cdot c)$</td>
<td>$O(m \cdot R^2 \cdot (c \cdot m + n^2))$</td>
</tr>
<tr>
<td>Execution</td>
<td>$O\left(\frac{(m+n-1)!}{(m-1)!} \cdot (n + c)\right)$</td>
<td>$O\left(\sqrt{\frac{n! \cdot n^m}{k}} \cdot (n \cdot \log_2(n) + c)\right)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(n + m + c)$</td>
<td>$O((n + m) \cdot \log_2(n))$</td>
<td>$O(m \cdot R^2 \cdot (c \cdot m + n^2))$</td>
</tr>
<tr>
<td>Code</td>
<td>$O(1)$</td>
<td>$O(n^2 \cdot c)$</td>
<td>$O(m \cdot R^2 \cdot (c \cdot m + n^2))$</td>
</tr>
</tbody>
</table>

$m$: number of machines  
$n$: number of transactions  
$c$: number of conflicts  
$R$: max. runtime  
$k$: number of solutions
Number of Solutions

| $|S_{R \leq l}|$ |
|----------------|
| $10^{11}$ |
| $10^{10}$ |
| $10^{9}$ |
| $10^{8}$ |
| $10^{7}$ |
| $10^{6}$ |
| $10^{5}$ |

$R$ as factor of $R_{opt}$

$m = 2, |O| = n \cdot 0\%$  
$m = 2, |O| = n \cdot 20\%$  
$m = 2, |O| = n \cdot 40\%$  
$m = 4, |O| = n \cdot 0\%$  
$m = 4, |O| = n \cdot 20\%$  
$m = 4, |O| = n \cdot 40\%$

$m = 2$  
$m = 4$

| $N$ | 8,589,934,592 | 2,199,023,255,552 |
| $k$ | 48,384,000 | 559,872 |
| $k$ for $\leq 1.25 \cdot R_{opt}$ | 1,472,567,040 | 2,047,306,752 |
Join Ordering with Quantum Annealing

- **Quantum Annealing** solves quadratic unconstrained binary optimization (QUBO) problems.

**Example Join Ordering:**

- $P$: power set of relations to join
- $P_k$: set of elements representing joins of $k$ relations, i.e.,
  
  $P_k = \{a|a \in P \land |a| = k\} \subseteq P$
- $w_{\text{max}} = \max(w_i) + c$, where $c > 0$ and $w_i$ represents the cost of join $i$
- **Rewarding joins with lowest costs:**
  
  $S_1 = \sum_{j=2}^{m} \sum_{i \in P_j} x_i \cdot (w_i - w_{\text{max}})$

**Minimize** $S = S_1 + S_2$

**Punish other combinations:**

$\Rightarrow S_2 = \sum_{i=2}^{m-1} \sum_{j=i}^{m} \sum_{y \in P_i} x_y \cdot x_z \cdot w_{\text{max}}$

**Constraint:**

$y \subseteq z$
Join Ordering with Quantum Annealing

- Real-world queries of the ErgastF1 Benchmark with PostgreSQL

- QPU access time approx. 100ms
QC4DB: Accelerating Relational Database Management Systems via Quantum Computing

- Project Website@Quantentechnologien 
- Project funded by BMBF
  - Duration 3 years, 1.8M Euros
- Topics
  - Query Optimization
  - Optimizing Transaction Schedules of an open source relational database management system
- Partners
  - University of Lübeck (Coordinator Sven Groppe)
    - Hardware-Acceleration of Databases
    - Website: https://www.ifis.uni-luebeck.de/~groppe/
  - Quantum Brilliance GmbH
    - Room Temperature Diamond Quantum Accelerators
    - Website: https://quantumbrilliance.com/
Summary & Conclusions

- Scheduling transactions as variant of jobshop problem with additionally considering blocking transactions
  - Hard combinatorial optimization problem $\Rightarrow$ hardware acceleration
- Enumeration of all possible transaction schedules for finding an optimal one
  - Hardware acceleration via quantum annealing
    - Formulating transaction schedule problem as quadratic unconstrained binary optimization (QUBO) problem
    - Constant execution time in contrast to simulated annealing on classical computers
    - Preprocessing time increasing with larger problem sizes
  - Grover's search: $\approx$ quadratic speedup on Universal Quantum Computers
    - Estimation of number of solutions for a further speedup
      - Estimation of speedup for suboptimal solutions being a guaranteed factor away from optimal solution
    - Code Generator available at [https://github.com/luposdate/OptimizingTransactionSchedulesWithSilq](https://github.com/luposdate/OptimizingTransactionSchedulesWithSilq)