HEX |||: SAP’s new HANA Execution Engine

Daniel Ritter
Cloud Database Architect & Member of HANA Research Campus at SAP SE

Sponsor talk, Conference on Innovative Data Systems Research (CIDR), 1/2023
HANA Core Engines and Stores

SQL / SQL Script

SQL Optimizer

Join Engine

OLAP Engine

Row Engine

Column Table

Row Table

Modular, disconnected

In-memory + disk (cf. NSE), HTAP (cf. Translytical Data Platform)

... and many more: e.g. Calculation engine (calculation views, star joins), MDS engine (multi-dimensional queries incl. aggregation, transformation, calculation), ...

Modular, new physical operators out of previous engines

Central, cross-optimizations

Composed into pipelines

Join Engine Operators
OLAP Engine Operators
Row Engine Operators

Column Table
Row Table
Overview

- HEX State-of-the-art engine for HTAP (see table)
- Workloads: transactional applications (e.g., S4/HANA), analytical queries (e.g., Data Warehouse Cloud)
- Data chunks
- JIT-L pipelined
- Data-centric code generation in L (LLVM convenience layer)
- L used also for, e.g., stored procedures
- Supportability: debugging, profiling L programs on tooling level; portability
- Extensible: New physical operators can be added to HEX (e.g., application- / service-specific)
- TCO, Price / Performance
  - Reduce memory footprint: pipelining and streaming, fewer engines (reduce intermediate result materialization)
  - More CPU-efficient due to JIT compilation
  - Performance same or slightly better
- Distributed query processing (send, receive)
- Intra pipeline parallelization (dynamic)

<table>
<thead>
<tr>
<th>Engine</th>
<th>Proc. Model</th>
<th>Data flow model</th>
<th>Level of Parallelism</th>
<th>Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>DuckDB</td>
<td>Vectorized</td>
<td>Pull (&quot;Vector Vulcano&quot;)</td>
<td>Intra (pipeline)</td>
<td>OLAP</td>
</tr>
<tr>
<td>HyPer / Umbra</td>
<td>JIT-LLVM / Pipelined</td>
<td>Push</td>
<td>Intra (pipeline), Inter?</td>
<td>HTAP</td>
</tr>
<tr>
<td>Hysire</td>
<td>Materialized (lazy)</td>
<td>Push</td>
<td>Intra (pipeline)</td>
<td>OLAP</td>
</tr>
<tr>
<td>Redshift</td>
<td>JIT-C++ / Pipelined + Vectorized?</td>
<td>Push</td>
<td>Intra (pipeline)</td>
<td>HTAP</td>
</tr>
<tr>
<td>HANA / HEX</td>
<td>JIT-L / Pipelined</td>
<td>Push</td>
<td>Intra (pipeline)</td>
<td>HTAP</td>
</tr>
</tbody>
</table>

Execution Phases

→ In practice
  → Works well / no issues for OLTP queries with plan caching
  → JIT compilation times challenging for large and complex analytical queries during cold start
→ Mitigate JIT compilation times
→ Start interpreted / uncompiled, compile in background per query / L program (fragment)
→ Switch to compilation after third execution
Example

```
SELECT A, A + B * 7 from X;
```

**Diagram Description:**
- **Table Scan** reads VIDs for A, B.
- **Read VIDs for A, B** reads values from dictionaries.
- **Send result** after projecting "A + B * 7".
- **Generated Operator** for Row in InputChunk:
  ```python
  for Row in InputChunk
  {
    vidA = row.column[0]
    vidB = row.column[1]
    valueA = dictA[vidA]
    valueB = dictB[vidB]
    expr = valueA + valueB * 7
    outRow = ...
  }
  ```
- **Fusion of operators**
- **Precompiled Operator**
- **Framework Generated Code**
- **Generated Operator**
- **Data Chunk**
Intra Pipeline Parallelization

- **Pipelining**: better memory access pattern (less cache misses) and no full materialization between operators (lower memory footprint)
- Parallelization with pipelining more complicated
  - Parallelize operators instead of data
  - Determining task size complicated: fixed task size → skewed workload
  - Parallelization requires (expensive) scheduling → bigger tasks sizes preferable, BUT due to skewed workloads → fine-grained tasks

**HEX Execution Framework**

- **Task Scheduler**
  - Add task definition
  - Schedule task
  - Create Worker Manager

- **Worker Manager (Job)**
  - Calculate #workers
  - Create Worker Jobs

- **Job Executor**
  - Call

- **Queue Manager**
  - Observes, maintains
  - Returns handle

- **Work Queue**
  - Stored task definitions, task priorities

- **Execution Flags** (Pause, Resume, Finish)

- **Worker Job**
  - Thread, processing tasks from Work Queue
Intra Pipeline Parallelization

- **Pipelining**: better memory access pattern (less cache misses) and no full materialization between operators (lower memory footprint)

- Parallelization with pipelining more complicated
  - Parallelize operators instead of data → determining task size complicated: fixed task size → skewed workload
  - Parallelization requires (expensive) scheduling → bigger tasks sizes preferable, **BUT** due to skewed workloads → fine-grained tasks

- Reduce / tame job creation overhead / scheduling:
  - HEX task scheduling integrated in HANA job scheduling
  - Map several tasks (possibly of different kind) to one job (pooled)
  - Job will live longer than task → less job creation overhead
Intra Pipeline Parallelization

- **Pipelining**: better memory access pattern (less cache misses) and no full materialization between operators (lower memory footprint)
- Parallelization with pipelining more complicated:
  - Parallelize operators instead of data → determining task size complicated: fixed task size → skewed workload
  - Parallelization requires (expensive) scheduling → bigger tasks sizes preferable, BUT due to skewed workloads → fine-grained tasks
- Reduce / tame job creation overhead / scheduling:
  - HEX task scheduling integrated in HANA job scheduling
  - Map several tasks (possibly of different kind) to one job (pooled)
  - Job will live longer than task → less job creation overhead
- Address workload skew: sampling / re-parallelization:
  - Worker Manager checks the Queue Manager regularly to calculate progress and creates more workers, if needed > #workers dynamic
  - Intermediate scheduling operators measure elapsed time to execute remaining pipeline (e.g., after selective / expanding joins, selective table scans) + find new, good task size
- Sampling not for free due to scheduling points:
  - Are sync. points > too many lead to fluctuations between runs
  - Break operator fusion

HEX Execution Framework

- Task Scheduler
  - Create Worker Manager
    - Schedule task
    - Add task definition
  - Create Worker Jobs
    - Schedule task
    - Calc. #workers
  - Call
  - Get next prio task

Worker Manager (Job)

- Queue Manager
  - Observes, maintains
  - Returns handle
  - Stored task definitions, task priorities

Job Executor

- Worker Job
  - Thread, processing tasks from Work Queue
  - Execution Flags (Pause, Resume, Finish)
Challenges and Opportunities

- Remove old engines “in-flight” without disruptions: no functional or performance regressions
- State-of-the-art, compiled, pipelined query engine with extensible architecture
- Multi-Model engines in HEX, nested file formats, ...

Join us later at CIDR:

- **Tuesday 4:50 pm**: **Data Pipes: Declarative Control over Data Movement** Lukas Vogel (Technische Universität München); Daniel Ritter (SAP); Danica Porobic (Oracle); Pinar Tozun (IT University of Copenhagen)*; Tianzheng Wang (Simon Fraser University); Alberto Lerner (University of Fribourg)
- **Wednesday 11:10 am**: **DASH: Asynchronous Hardware Data Processing Services** Norman May (SAP SE)*; Daniel Ritter (SAP); Andre Dossinger (SAP SE); Christian Faerber (Intel Corporation); Suleyman Demirsoy (Intel Corporation)

Special thanks go to our academic and industrial collaboration partners as part of the SAP HANA Research Campus!

Ph.D. position available!
Thank you!

Contact information:

Daniel Ritter
E-Mail: daniel.ritter@sap.com