SAP HANA: A Data Platform for Enterprise Applications Purpose Built for Modern Hardware

Anil K Goel, SAP Canada

May 1, 2019
Agenda

• Recap: Traditional Enterprise Data Management Architectures

• Vision: Enterprise Operational Analytics Data Management Systems

• Reality: Design and Implementation Aspects of SAP HANA
  • Key Design Principles and Concepts
  • Focus: The new HANA EXecution Engine (HEX)
  • Focus: Storage Extensions

• Conclusion
About SAP

SAP is the **world leader in enterprise applications** in terms of software and software-related service revenue. Based on market capitalization, we are the world’s **third largest independent software manufacturer**.

- **437,000+** Customers in more than 180 countries
- **98,500+** Employees in 144+ countries (03-31-2019)
- **€24.74bn** Total Revenue (non-IFRS) in FY2018
- **92%** Of Forbes Global 200 are SAP customers
- **47 yrs.** Of history and innovation
- **100+** Innovation and development centers
- **19,200+** SAP partner companies globally
- **~195 mil.** Users in SAP cloud user base
Global Database Development Team
SAP Labs Waterloo

Established Database R&D site going back 35+ years

~220 people

- Majority SAP Database Research and Development
- Work on HANA, Cloud-native databases, distributed databases, Edge Computing

Co-op & Grad Student intern programs

Strong academic collaborations

We are hiring!

- Full-time positions
- Student jobs - master/bachelor theses, graduate internships

http://www.careersatsap.com/
http://jobs.sap.com
https://www.saphana.com
Traditional Enterprise Data Management Architectures

Transactional System for Online Transaction Processing (OLTP)
• Short-running, trivial statements; frequent updates; high amount of concurrent users

Data Warehouse for Online Analytical Processing (OLAP)
• Long-running, complex statements; (almost) read-only; few users

Nightly Data Extraction, Transfer, and Loading Between Systems (ETL)
• Transformation from OLTP-friendly to OLAP-friendly format; indexing; pre-aggregation
Challenges of Traditional Architectures

High Costs / TCO
- Two (different?) database management systems
- Two times the license costs
- Two times the hardware
- Two times the administration

No Data Freshness for Reporting
- Data is updated only nightly
- Reporting during the day always sees stale data

ETL Processes are a Pain
- Complex to design and maintain
- Need to complete in daily maintenance window (e.g. <8 hours)
- Data preparation necessary for reporting
- Slow reaction to changing business demands (no ad-hoc reporting)
From Vision To Reality: Objectives for HANA

• Good Enough for OLTP
  • Unrealistic to beat a specialized, pure OLTP engine
  • But: Be able to sustain a typical enterprise workload
  • Example: 40,000 SQL statements / second

• Excel in Analytics
  • Flexible reporting without pre-computation / aggregates
  • Leverage modern hardware (multicore + large DRAM capacity)
  • Marketing: „Subsecond everything“
  • Vision: „window of opportunity“

• Support both OLTP and OLAP in ONE database
Key Design and Implementation Aspects of SAP HANA
SAP HANA Database
Multi-Engine for Multimodal Enterprise Applications

Business Applications

Connection and Session Management
- SQL
- SQL Script
- MDX
- Graphs
- Planning

Authorization Manager
- Calculation Engine
- Optimizer and Plan Generator

Execution Engine

In-Memory Processing Engines
- Relational Engine
- Graph Engine
- Text Engine

Metadata Manager

Persistency Layer
- Logging and Recovery
- Storage Management

© 2019 SAP SE or an SAP affiliate company. All rights reserved.
Distributed Share-Nothing In-Memory Computing
Observation: Enterprise Workloads are Read Dominated

- Workload in Enterprise Applications consists of:
  - Mainly read queries (OLTP 83%, OLAP 94%)
  - Many queries access large sets of data
SAP HANA Column Store Main: Fast Queries

- Read-Optimized, Immutable Data Store

- Dictionary Compression
  - All data in columnar tables is dictionary compressed
  - Dictionary is prefix-compressed
  - Dictionary is sorted in same order as data values

- Efficient secondary data compression (run-length, cluster, prefix, etc.)
  - Heuristic algorithm orders data to maximize compression of columns

- Compression schemes work well, e.g.,
  - Speeding up operations on columns to factor 10
  - Reduces Storage up to factor 5 for typical SAP data schemes

<table>
<thead>
<tr>
<th>pos</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Adam</td>
</tr>
<tr>
<td>1</td>
<td>Adriana</td>
</tr>
<tr>
<td>2</td>
<td>Alexa</td>
</tr>
<tr>
<td>3</td>
<td>Amber</td>
</tr>
</tbody>
</table>

Indexvector

<table>
<thead>
<tr>
<th>pos</th>
<th>valueID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>
SAP HANA Column Store Delta: Write Optimized Update Support

- Write-Enabled Table Fragments Handle all Updates
  - Only update operation on main is to delete rows
  - UPDATEs modelled as DELETE+INSERT
  - INSERT append to delta store

- Dictionary not Sorted
  - No need to recode column vectors upon delete/insert

- Additional B-Tree for Efficient Lookup
  - Allows to quickly retrieve valueID for value
  - Essential for fast unique checks upon insert
  - Can be used for range queries
  - Less compression of data
  - Delta is merged with main periodically, or when thresholds exceeded
    - Delta merge for a table partition is done on-line, in background
    - Enables highly efficient scan of Main again

<table>
<thead>
<tr>
<th>pos</th>
<th>valueID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pos</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Keith</td>
</tr>
<tr>
<td>1</td>
<td>Adriana</td>
</tr>
<tr>
<td>2</td>
<td>Roland</td>
</tr>
<tr>
<td>3</td>
<td>Bert</td>
</tr>
</tbody>
</table>

Dictionary

B-Tree
Delta Merge

- Consolidation of Delta and Main into new Main
  - Improves query performance (especially for analytics)
  - Reduces memory footprint (no B-Tree for dictionary necessary)

- Automatically triggered by the System based on Cost-Based Decision Function
  - Considers delta:main ratio, size in RAM and disk, system workload
  - Performed on a per table-basis (actually: partition-based), parallelized on column-level
### SAP HANA Technology

**Compression with run length encoding**

#### Classical Row Store

<table>
<thead>
<tr>
<th>Company</th>
<th>Region</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEL</td>
<td>USA</td>
<td>A</td>
</tr>
<tr>
<td>Siemens</td>
<td>Europe</td>
<td>B</td>
</tr>
<tr>
<td>Siemens</td>
<td>Europe</td>
<td>C</td>
</tr>
<tr>
<td>SAP</td>
<td>Europe</td>
<td>A</td>
</tr>
<tr>
<td>SAP</td>
<td>Europe</td>
<td>A</td>
</tr>
<tr>
<td>IBM</td>
<td>USA</td>
<td>A</td>
</tr>
</tbody>
</table>

#### HANA Column Store: Dictionary compressed

<table>
<thead>
<tr>
<th>Company</th>
<th>Region</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEL</td>
<td>Europe</td>
<td>A</td>
</tr>
<tr>
<td>Siemens</td>
<td>USA</td>
<td>B</td>
</tr>
<tr>
<td>SAP</td>
<td>USA</td>
<td>C</td>
</tr>
</tbody>
</table>

#### HANA Column Store: Run length compressed

<table>
<thead>
<tr>
<th>Company</th>
<th>Region</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEL</td>
<td>Europe</td>
<td>A</td>
</tr>
<tr>
<td>Siemens</td>
<td>USA</td>
<td>B</td>
</tr>
<tr>
<td>SAP</td>
<td>USA</td>
<td>C</td>
</tr>
</tbody>
</table>

*Note that there is a variety of compression methods and algorithms like run-length compression.*
Single Instruction Multiple Data (SIMD)

Scalar processing
– traditional mode
– one instruction produces one result

SIMD processing
– with Intel® SSE & AVX
– one instruction produces multiple results

Scalar processing diagram:
- SOURCE
- Scalar OP
- X
- Y
- DEST
- XopY

SIMD processing diagram:
- SOURCE
- SSE/2/3 OP
- X4, X3, X2, X1
- DEST
- X4opY4, X3opY3, X2opY2, X1opY1
- 127
- 0
Technical Deep Dive

The New HANA EXecution Engine (HEX)
Status Quo: HANA Query Processing Engines

- Operate on dictionary-compressed data representations
- Designed with analytics in mind
  - Also: Optimized and tuned for OLTP
  - Example: Full semi-join reduction for all joins
HANA as a general purpose database: Novel focus areas

- Already excellent performance
- New hot spot: Short-running queries
  - Allow migration of existing / legacy application **without code changes**
  - Many applications are not DBMS friendly
- New hot spot: Memory footprint reduction
  - HANA materializes intermediate results between operators
  - Good strategy for analytics, but high intermediate memory consumption
  - Alternative: Pipelined execution

General Purpose Database
Next generation query processing in HANA

**Goal:** Extend HANA with a next-generation execution engine

- Reduce runtime memory footprint
- Replace „old“ engines to avoid complexity increase
- Maintain competitive advantage
- Exploit columnar and dictionary-compressed storage
- Operate on compressed data representations
- Incorporate latest research results: Code generation
- Academic frontrunner: TU Munich / HyPer
- Commercial systems: SAP HANA Vora/Velocity, Microsoft Hekaton

**Efficiently Compiling Efficient Query Plans for Modern Hardware**

Thomas Neumann
Technische Universität München
Munich, Germany
neumann@lein.tum.de
Code generation in HEX

- HEX uses LLVM for machine code generation
- Does not generate LLVM intermediate representation (IR) directly
- Uses SAP‘s „L“ language frontend instead
  - L is already used within HANA, e.g. for stored procedures
  - Code is easier to read than IR
  - Benefit from existing infrastructure (supportability, debugging, profiling…)
  - L already contains implementations of SQL expression functions (add_years(), …)
Compilation Times

- To mitigate, HEX makes use of the Interpreter provided by L
- Each piece of code is executed with the interpreter at first
  => Heavy performance penalty, but execution can start immediately
- When executed often: trigger compilation via LLVM (asynchronously)
- Once finished, execution switches to the compiled version

L interpreter is not optimal yet
- Interprets L program (instructions)
- Optimized bytecode interpreter provides much better performance
- See recent work from TU Munich

The Dark Side of code generation
Additional challenges

- Materialization strategies (early vs late)
  - Currently: Late materialization to keep intermediate results small
  - Research: Always best approach?

- Tuple-at-a-time vs blockwise processing (goal: exploit SIMD instructions in complete pipeline)
  - Trade-off with register locality of tuples

- Finding the right mix of compilation / interpretation for very complex query plans
  - Research results (TUM): Compilation does not finish for extremely complex, generated queries

- …
Technical Deep Dive

HANA Native Storage Extensions: Adding disk processing to an in-memory database server
Idea: load pieces of a column

- Not every column of each table needs to be in memory
- Not all rows of a column need to be in memory
- Not all data structures of a column need to be in memory

- Load portions of data vectors needed for a query
- Load portions of dictionaries needed for search
- Load portions of inverted indices corresponding to query predicates

Page in/out read optimized portion:

- More memory consumed by read only portion of a column
Native Storage Extension (VLDB 2019)

Value proposition:

- Increase HANA data capacity at low TCO
- Deeply integrated warm data tier, with full HANA functionality
- Support for all HANA data types and data models
- Simple system landscape
- Scalable with good performance
- Supported for both HANA on-premise and HANA-as-a-Service (HaaS)
- Available for any HANA application
- Complements, without replacing, other warm data tiering solutions (extension nodes, dynamic tiering)

Use cases:

- Any customer built or SAP built HANA application that is challenged by growing data volumes
- S/4HANA and Suite on HANA data aging framework (replace current “page loadable columns” solution)
- Reduced storage size and TCO for cloud deployments of internal systems and HaaS applications
- Expand capacity in HANA extension nodes (BW use case being evaluated)
Unified Persistence
Compatible with Memory and Paged Primitives

Hybrid Column Structures
Dictionary, Data Vectors, Indexes
May be loaded in memory or paged

Hybrid Column Store
Specialized Column Implementations:
Uncompressed, Compressed, Spatial, RowID, HashKey

Memory Primitives

Paged Primitives

Common primitives, e.g., SIMD scans

Hex Query Engine

Partition Management

Elastic Buffer Cache

NSE Configuration DDLs

NSE Advisor

Data Temperature Statistics
NSE Recommendation Engine

- Data access frequency collected at query level for each column (scan count etc.) to indicate data hotness
- Rule based heuristics to identify cold objects (column, partition, table) with large memory footprint as page loadable candidates
- User adjustable threshold for cold and hot objects
- Flexible interface for user to act on individual recommended object to convert from column loadable to page loadable or vice versa.
Wrapping up
Conclusion

- **Operational Analytics DBMS are a Game-Changer for Enterprise Applications**
  - Fast reporting even without indexes / aggregates / materialized views (CRM)
  - Additional insights from reporting on transactional data (HANA Live / Suite on HANA)
  - Complete application redesign (S4HANA)
    - No aggregates (simplification)
    - Code pushdown
    - Fast analytics on current data in real-time
Challenges and Next Steps

Cloud, Memory Footprint, and Hardware Costs
Performance Challenged by Specialized Engines and NoSQL
Towards Polyglot Persistency / Data Management Platforms

CMU PDL Tech Talk: May 2, 2019, 12:00pm
Anil Goal, SAP Canada

Anil.Goel@sap.com