In-Memory Data Management for Enterprise Applications

BigSys 2014, Stuttgart, September 2014
Johannes Wust
Hasso Plattner Institute (now with SAP)
What is an In-Memory Database?

I. INTRODUCTION

In a main memory database system (MMDB) data resides permanently in main physical memory; in a conventional database system (DRDB) it is disk resident. In a DRDB, disk data may be cached into memory for access; in a MMDB the memory resident data may have a backup copy on disk. So in both cases, a given object can have copies both in memory and on disk. The key difference is that in MMDB the primary copy lives permanently in memory, and this has important implications (to be discussed) as to how it is structured and accessed.

A Combination of Hardware and Software Innovations Makes In-Memory Data Management Feasible

- Up to 4 TB DRAM in commodity servers
- Large data sets in-memory
- Multi-core architecture
- Massively parallel execution
- In-Memory row and column Store
- Column = Fast queries
- Compression
- 5 – 20x
- Main-Delta architecture
- Fast inserts
- No aggregate tables
- ...
Two Different Principles of Physical Data Storage: Row vs. Column Store

<table>
<thead>
<tr>
<th>Document Number</th>
<th>Document Date</th>
<th>Sold-To Party</th>
<th>Order Value</th>
<th>Status</th>
<th>Sales Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>95769214</td>
<td>2009-10-01</td>
<td>584</td>
<td>10.24</td>
<td>CLOSED</td>
<td>Germany Frankfurt</td>
</tr>
<tr>
<td>95769215</td>
<td>2009-10-01</td>
<td>1215</td>
<td>124.35</td>
<td>CLOSED</td>
<td>Germany Berlin</td>
</tr>
<tr>
<td>95779216</td>
<td>2009-10-21</td>
<td>584</td>
<td>47.11</td>
<td>OPEN</td>
<td>Germany Berlin</td>
</tr>
<tr>
<td>95779217</td>
<td>2009-10-21</td>
<td>454</td>
<td>21.20</td>
<td>OPEN</td>
<td>Germany Frankfurt</td>
</tr>
</tbody>
</table>

**Row Store**

**Column Store**
OLTP vs. OLAP Queries Favor Different Storage Patterns

SELECT * 
FROM Sales Orders 
WHERE 
    Document Number = '95779216'

SELECT SUM(Order Value) 
FROM Sales Orders 
WHERE Document Date > 2009-01-20
Enterprise Data Stored in Columns Can be Compressed by a Factor of 5-10

<table>
<thead>
<tr>
<th>Document Number</th>
<th>Document Date</th>
<th>Sold-To Party</th>
<th>Order Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>95769214</td>
<td>2009-10-01</td>
<td>584</td>
<td>10.24</td>
</tr>
<tr>
<td>95769215</td>
<td>2009-10-01</td>
<td>1215</td>
<td>124.35</td>
</tr>
<tr>
<td>95779216</td>
<td>2009-10-21</td>
<td>584</td>
<td>47.11</td>
</tr>
<tr>
<td>95779217</td>
<td>2009-10-21</td>
<td>454</td>
<td>21.20</td>
</tr>
</tbody>
</table>

Dictionaries

- Typical compression factor for enterprise software 5-10
- In financial applications up to 50
A Main–Delta Architecture ensures Fast Inserts

Insert

Periodic Merge
Removal of Materialized Aggregates Dramatically Decreases Complexity (1/7)

The prime example from about every database text book:

*Wire money from Account A to Account B*

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>5</td>
</tr>
<tr>
<td>Bob</td>
<td>25</td>
</tr>
</tbody>
</table>

**Start Transaction**

Set Balance=Balance+7 where Account=Alice;
Set Balance=Balance-7 where Account=Bob;

**Commit Transaction**
Every real-world application that deals with money needs to be auditable.

### Account Balances

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>12</td>
</tr>
<tr>
<td>Bob</td>
<td>18</td>
</tr>
</tbody>
</table>

### Account Transactions

<table>
<thead>
<tr>
<th>Account</th>
<th>Date</th>
<th>Account Type</th>
<th>Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>1</td>
<td>Supplier</td>
<td>+5</td>
</tr>
<tr>
<td>Bob</td>
<td>1</td>
<td>Customer</td>
<td>+22</td>
</tr>
<tr>
<td>Bob</td>
<td>2</td>
<td>Customer</td>
<td>+3</td>
</tr>
<tr>
<td>Alice</td>
<td>5</td>
<td>Supplier</td>
<td>+7</td>
</tr>
<tr>
<td>Bob</td>
<td>5</td>
<td>Customer</td>
<td>-7</td>
</tr>
</tbody>
</table>

Lots of work to keep balances and accounts consistent:
- Each logical insert becomes two (or more)
  → Extra storage space necessary (Disks, Backup...)
- If two concurrent transactions affect same account: conflict/rollback.

Summing up all entries for an account is too expensive in a row store to be done at runtime. Periodically updating materialized aggregates therefore was the only way to do reporting in the past.
Removal of Materialized Aggregates Dramatically Decreases Complexity (3/7)
Removal of Materialized Aggregates Dramatically Decreases Complexity (4/7)
Removal of Materialized Aggregates Dramatically Decreases Complexity (5/7)
Removal of Materialized Aggregates Dramatically Decreases Complexity (6/7)
Removal of Materialized Aggregates Dramatically Decreases Complexity (7/7)

Classic SAP Financials with Transaction-Maintained Aggregates

Simplified SAP Financials
Dramatic Performance Increase Enables Reuniting Transactional and Analytical Data Processing

- No up-to-date data for analytics
- Redundant data leads to
  - Increased storage
  - Inconsistencies
- Enables new “mixed workload” applications
- Reduces system complexity
- Allows simplifications due to removal of aggregates
Mixed Workload Management

Challenges

• Different query complexity
• Different query execution (single-threaded vs. parallel)
• Different service level expectation

Mixed Workload Applications (eg. interactive analytics) (OLXP)

Transactional applications (OLTP)

Analytical applications (OLAP)

In-Memory Enterprise Database
Query Processing on Multicores (1/2)

Query Compilation

- Query Parsing
- Logical Plan optimization
- Selection of algorithms
- Result materialization
- Choose degree of parallelism (potentially online)

select left.VBELN from VBAP as left and VBAP as right where left.VBELN = right.VBELN, left.MATNR = (..), right.MATNR = (..)
Query Processing on Multicores (2/2)

Query Execution

q2 q3 q3 q1 q2

Challenges for Workload Management

- Degree of Parallelism
- Resource Assignment
- Underutilization
- Overcommitment
- Space Shared Resources
- Non-uniform memory access
- Sharing intermediate results
A Task-based Execution Model forms a Basis for Effective Workload Management for IMDBMS

**Query Stream**
- q2
- q3
- q3
- q1
- q2

**Task Queue**

**General approach**
- Transform query plans into tasks and schedule tasks on worker threads
- Number of worker threads equals number of HW contexts
- Apply workload management by queue policies

**Advantages:**
- Reduced overhead of thread scheduling
- Good load balancing: effective degree of parallelism of queries varies with load
- A “common denominator” for scheduling: Interleaved execution of short and long running queries
Query Plans are Transformed to Task Graphs by Replacing Partitionable Operators with Data Parallel Tasks

Query Transformation

Replace all partitionable operators by
- A number of data parallel tasks
- A task to combine results
Main Findings/Contributions for Workload Management with a Task-based Query Model

1. **Priority of query classes**: A non-preemptive priority scheduling policy provides an effective way to prioritize query classes.

2. **Impact of task granularity**: The granularity of tasks impacts throughput and responsiveness of a loaded system.
   - Coarse granular tasks increase throughput of OLAP (up to 30%).
   - Fine granular tasks increase responsiveness of short running queries (factor >9).

3. **Limit maximum task sizes**: A system-wide maximum task size (MTS) effectively balances mutual performance impact of query classes.

4. **Enforce predefined resource shares**: Dynamic task priorities allow enforcing predefined resource shares for sessions.

5. **Resource aware scheduling**: By pinning threads to nodes, we can leverage knowledge about resource consumption of tasks to reduce contention (up to 25% runtime reduction).
**1 Priority of query classes**

- **Experiment setup:**
  - 3 Query classes, OLTP, OLXP, OLAP
  - 1 OLTP user, 1 OLXP user with 1s think time, 1-128 OLAP user
  - A non-preemptive priority scheduling policy shortens the wait time in the queue for high priority tasks
    - Priorities are set by application, by user or by query type
    - Per socket / per thread priorities for scalability (No guarantee that highest priority task is executed next)

Test machine: 4 Intel(R) Xeon® X7560 CPUs, 8 cores each and 512GB RAM
In a loaded system, a low degree of parallelism for OLAP ...

- ... leads to larger task sizes for OLAP queries
- ... reduces overhead and increases OLAP throughput

- ... increases response time for high priority tasks, due to non-preemptive queueing policy
3 Limit maximum task sizes (1/2)

- How to enforce maximum task size?
- Estimate sequential runtime of operator when inputs are known
- Simpler problem than predicting query execution times
- Runtime estimation
  - Model based on $r_{\text{par}}(n, t) = r_{\text{seq}}(t)/n + ov(n)$
  - Calibrate for given machine using weighted nonlinear least-square fitting
Enforce predefined resource shares (1/3)

- Constant stream of queries with different runtime
- Share of DBMS execution time converges to runtime of queries

Test machine: 2 Intel(R) 5670 CPUs with 6 cores each and 144GB RAM.
Motivation: enforce a fair/defined share of resource usage for database sessions

Idea: dynamically monitor execution time and adjust priorities to converge to a predefined share

The problem of shared query execution: minimize the overall deviation of the work share from the target share over an interval $T$:

$$\Delta S = \int_0^T \sum_{s_i \in S} |ts_i - ws_i(t)|$$

We provide an approximate solution based on calculating the moving average with a window size $n$:

$$ws_i(t) = \frac{1}{n} \sum_{\{t-n,\ldots,t\}} \frac{w_i(t)}{W(t)}$$
Enforce predefined resource shares (3/3)

3 Sessions with 3 analytical queries, largest table 10m records
- Session 1, 3 user, start t=0s, target share = 25%
- Session 2, 3 user, start t= 30s, target share = 50%
- Session 3, 6 user, start t = 120s, target share = 25%

Test machine: 2 Intel(R) 5670 CPUs with 6 cores each and 144GB RAM.
Conclusions

• **In-Memory Data Management for Enterprise Applications is feasible due to a combination of** hardware developments and database design decisions.

• Applying In-Memory Data Management to Enterprise Applications **allows a simplification of data models and system landscapes, and enables entirely new types of Applications**.

• **Managing mixed application workloads on In-Memory databases is a challenge** when running transactional and analytical applications on a single database instance.

• **A task-based query execution model** is suitable to enforce workload management policies while maintaining the performance of in-memory computing.