Adaptive Query Processing on Prefix Trees
Wolfgang Lehner

Fachgruppentreffen, 22.11.2012 – TU München
Challenges for Database Systems

“Three things are important in the database world: performance, performance and performance.”

Bruce Lindsey

The DBMS Landscape – Performance Needs

Extreme data

Extreme performance
Challenges for Database Systems

- Extreme Performance
- Extreme Flexibility
- Extreme Data

Flexibility in Database Systems

+ during deployment time (schema definition)
- during database lifetime (schema evolution)
- during query runtime (scheduling, …)
> Flexibility from 10.000 feet

Database Technology Group

Data comes first, schema comes second

Querying Web-Tables
Role-based object models

Open Data platforms

Demand flexibility

- Alternative query model (drill-beyond)
- Domain-specific data models
- Statistical operators

Data model-related

Storage model-related

Provide flexibility

- Record management
- Logging an persistency
- HW/SW DB-CoDesign
- (MV)CC

Query processing

Database System

- Flexibility is cross-cutting

Operating system & hardware

Applications

- Operating system & hardware
- Flexibility is cross-cutting
### Example 1: Data-driven Schema Design

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GSM</strong></td>
<td>GSM</td>
<td>GSM TriBand</td>
<td>GSM, GPRS, Bluetooth, FM</td>
<td>GSM, UMTS, Bluetooth, USB, WiFi</td>
<td>GSM, UMTS, HSPA+, Bluetooth, FM, USB, WiFi, aGPS, HDMI</td>
</tr>
<tr>
<td><strong>Dimensions/Weight</strong></td>
<td>Dimensions/Weight/Form Factor</td>
<td>Dimensions/Weight/Form Factor</td>
<td>Dimensions/Weight</td>
<td>Dimensions/Weight</td>
<td>Dimensions/Weight</td>
</tr>
<tr>
<td><strong>DotMatrix Display</strong></td>
<td>DotMatrix Display</td>
<td>2.0”, QVGA, Color</td>
<td>3.5”, 320×480, Touch</td>
<td>4.3”, 800×480, Touch</td>
<td></td>
</tr>
<tr>
<td><strong>Standby/Talk Time</strong></td>
<td>Standby/Talk Time</td>
<td>Standby/Talk Time</td>
<td>Battery Capacity</td>
<td>Battery Capacity</td>
<td></td>
</tr>
<tr>
<td><strong>#Names, #SMS</strong></td>
<td>#Names, #SMS</td>
<td>Memory/Card Slot</td>
<td>Memory</td>
<td>Memory/Storage/Card Slot</td>
<td></td>
</tr>
<tr>
<td><strong>Polyphonic Ringtones</strong></td>
<td>Polyphonic Ringtones</td>
<td>MP3 Ringtones/MP3 Player/Browser</td>
<td>Browser/Exchange</td>
<td>Browser/Exchange</td>
<td></td>
</tr>
<tr>
<td><strong>2.0”, QVGA, Color</strong></td>
<td></td>
<td>3MP Camera</td>
<td>Front/Rear Camera/HD Video</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>3.5”, 320×480, Touch</strong></td>
<td></td>
<td>3MP Camera</td>
<td>CPU, OS</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>4.3”, 800×480, Touch</strong></td>
<td></td>
<td></td>
<td>Adaptive Query Processing on Prefix Trees</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Prof. Dr.-Ing. Wolfgang Lehner | Technische Universität Dresden
Example 2: Web Data Management

Web Data: A Multitude of Data Models and Sources

- Example 1: Queries spanning millions of data sources with as many schemata
- Example 2: Augment local datasets with (external) third-party information

select ... from cust, where nations.GDP > $1B
Example 3: Energy-Adaptive High-Speed Computing Platform

Optical Interconnect
- adaptive analog/digital circuits for e/o transceiver
- embedded polymer waveguide
- packaging technologies (e.g. 3D stacking of Si/III-V hybrids)
- 90° coupling of laser

Radio Interconnect
- on-chip/on-package antenna arrays
- analog/digital beamsteering and interference minimization
- 100Gb/s
- 100-300GHz channel
- 3D routing & flow management
QPPT Overview

Composed Operators
- Construction of composed operators, like multi-way-select-join-group-sort
- Avoids Intermediate Materialization

Cooperative Operators
- Operators optimize their output for the next operator

Intermediate Indexed Tables
- Operators exchange clustered indexes

Prefix Tree-based Indexes

QPPT: Query Processing on Prefix Trees

Thomas Kissinger, Benjamin Schlegel, Dirk Hebic, Wolfgang Lehner
Database Technology Group
Technische Universität Dresden
01062 Dresden, Germany
(firstname.lastname@tu-dresden.de)

ABSTRACT
Modern database systems have to process huge amounts of data and should provide results with low latency at the same time. To achieve this, data is nowadays typically held completely in main memory, to benefit of its high bandwidth and low access latency that could never be matched with disks. Current in-memory databases use in-memory tables, where exchange columns or vectors between operators and suffer from a high tuple reconstruction overhead. In this paper, we present the novel adaptive prefix processing model that makes indices the first-class citizens of the database system. The processing model comprises the concepts of adaptive indexed tables and cooperative operators, which make indices the common data exchange format between plan operators. To keep the intermediate index materialization cost low, we imagine optimized index trees that can be chosen by the optimizer. The prefix processing model allows the construction composed operators like the multi-exchange-join-optimizing such operators speed up the processing of complex SQL queries so that our approach surpasses static clustered index databases.

In-Memory Database

Cluster Indexes
Prefix Trees

- Unbalanced
- Balanced read/write performance
- Main memory-optimized
- Efficient parallel access

B+-Trees

- Balanced
  - Low update performance
  - Disk-optimized
  - Low scalability
Index Landscape

- KISS-Tree → 32bit key index
- Buzzard → NUMA optimized

Clustered Column Store → Scan optimized

Adaptive Radix Tree (ART)
KISS-Tree Overview

Properties
- Specialized version for 32bit keys
- Latch-free updates
- Order-preserving
- 2-3 memory accesses per key

→ Comparable fast to reported order-preserving in-memory indexes for read access
→ BUT:
  High update performance

- Heterogeneous in-memory index structure
- Combination of direct and indirect addressing
- Takes advantage of virtual memory management
- Enables different compression mechanisms
Level 1: Virtual Level

Decimal Key: 327914

Split key in 3 fragments

- Calculate corresponding 2nd node address from the first fragment
  - Direct addressing
  - No memory required for level 1

Requirement for Level 2: All nodes have to be stored sequentially in memory
> Level 1: Virtual Level

Decimal Key: 327914

- 16bit ($f_1$)
- 10bit ($f_2$)
- 6bit ($f_3$)

Split key in 3 fragments

Virtual Memory Address

Level 2

4 KB 4 KB 4 KB 4 KB 4 KB 4 KB ...

Memory Management Unit (MMU)

Page Directory

Physical Memory Address

Operating System

Hardware

© Prof. Dr.-Ing. Wolfgang Lehner |
Level 2: On-demand Level

Change over to indirect addressing
- 1024 buckets per node containing a compact pointer to the 3rd level node
- 256 MB maximum

Virtual Memory Address

- $2^{16} + 2^{10} = \text{potential L3 leaf nodes of a size between } 2^0 \text{ and } 2^6$
Level 2: On-demand Level

- Allocation of consecutive virtual memory segments for each of the 64 node sizes possible on the third level
- Each segment consists of a maximum of $2^{26}$ blocks of the respective node size

Number of existing elements in L3 leaf node (compression)

Position of L3 leaf node in corresponding memory segment

$\Rightarrow 2^{26}$ blocks of size 1 pre-allocated

$\Rightarrow 2^{26}$ blocks of node size 64 pre-allocated
Level 3: Compression

Decimal Key: 327914

- 16bit ($f_1$)
- 10bit ($f_2$)
- 6bit ($f_3$)

- 64 possible node sizes
- Bitmap indicates which bucket is in use
- Only existing values are stored

64bit Bitmask

- Read-copy-updates
- Compare-and-swap
  → No in-place updates possible (lost updates)

Thread-Local Memory Management Subsystem
Duplicate Handling

- Efficient duplicate handling necessary for query processing
  - Scanning a linked list results in random memory accesses

- Page boundaries are a barrier for hardware prefetchers
  - store values sequentially in 4KB blocks
  - blocks grow exponentially until reaching 4KB
  - trade-off between scan performance and memory consumption
Prefix Tree Performance

Insert/Update Performance

- Performance depends on the number of memory accesses per key
  - KISS-Tree: 3 memory accesses per key
- Latency Hiding with Batch Updates/Lookups
- KISS-Tree performs better than hash table implementations
Cooperative Operators

- Each operator adjusts its output to the requirements of the successive operator

- traditional operators mostly build internal indexes anyway
  - skip plain tuple exchange overhead
  - provide high-quality input to each operator
Composed Operators (1)

- Materialization is costly; thus try to completely avoid it!

- Integrate operators that materialize large intermediate results.
Composed Operators (2)

- Materialization is costly; thus try to completely avoid it!

- 3-way-select-join

```cpp
class FILTERPT4: public PlanOperator {
  ...
  void joinWith(int joinnum, int outputnum, u_int64_t** buffer, int bufferpos, bool final);
  u_int64_t**** joinbuffers;
  ...
}
```
Evaluation (1)

- Star Schema Benchmark (SSB)
  - Scale Factor = 15

- Intel i7-3960X
  - 15 MB LLC
  - 3.3 GHz (Turbo Mode disabled)
- 32 GB Main Memory
- Ubuntu 12.04
- Single-threaded execution
Evaluation (2)

(*) : we are in touch with MonetDB; performance numbers are under investigation
Evaluation (3)

- SSB Query 1.1
- Single Join
- Select-Join operator avoids intermediate result materialization
- Batched KISS-Tree Lookups/Inserts

SSB Query 4.1
- 4 Joins over 5 Tables
- Multi-way join operator avoids intermediate result materialization
Adaptive QPPT (1)

- Flexible Query Execution on Adaptive Hardware
Adaptive QPPT (2)

- reduce hops by wireless „point-to-point“ interconnects
- operator → data / data → operator shipping
- adaptive hardware components (e.g., trade cache for compute units)
Comprehensive Approach: 9 Paths

- Devices & Circuits
  - CMOS (industry focus)
  - A Silicon Nanowires
  - B Carbon
  - C Organic
  - D Biomolecular Assembly
  - E Chemical
- Information Processing
  - H HAEC: Highly Adaptive Energy-Efficient Computing
- Functions
  - F Orchestration
  - G Resilience

"More-Shots-on-Goal"
Adaptive Query Processing on Prefix Trees
Wolfgang Lehner