AMAxoS—Abstract Machine for Xcerpt

1. Principles
2. Architecture

François Bry, Tim Furche, Benedikt Linse

PPSWR ’06, Budva, Montenegro, June 11th, 2006
Abstract Machine(s)

Definition and Variants …

- abstract machine := interpreter for low-level code
  - according to “machine” model (representation, instruct. set)
- abstract machine ~ virtual machine
- why abstract machines?
  - thought models
  - hardware or OS-level virtualization
  - AMs for high-level (programming) languages
Virtualization everywhere ...

OS-level

Intel Virtualization Technology

Once confined to special-purpose hardware, Intel Virtualization Technology is now supported in all modern Intel processors, allowing for processor virtualization in a variety of operating systems.

Transmeta Efficeon

The Transmeta Efficeon is a software hierarchy that allows for efficient virtualization of hardware resources.

Carbon Basics

This chapter describes the basics of Carbon and how it fits into Mac OS X, including the tools you use to build Carbon applications.

What's in Carbon?

© 2005 Apple Computer, Inc. All Rights Reserved.

Chapter 1
Carbon Basics

© 2002 http://www.jinwicked.com

© 2005 Apple Computer, Inc. All Rights Reserved.

Common Language Infrastructure (CLI) Partitions I to VI

Standard ECMA-335
**AM\(\chi\)oS—AM for Web Querying**

Operational Semantics for Xcerpt

- **abstract machine** ~ instruction set + machine model
  - just like algebra ~ operators + data model

- **both: precise query semantics**
  - on a “logical” level ——— on the operational/physical level

- **“optimizability”**
  - different combinations of instructions with *equivalent* overall result but different *performance* characteristics
Language neutral
- starting with a bias towards Xcerpt
- already: query core applicable for \{Xcerpt, XQuery, XSLT, SPARQL\}

Focus: in-memory processing of distributed data
- no (guaranteed) control over storage and indexing of data
- ad-hoc index creation (like XSLT key) and data model selection

+: distributed evaluation (if additional query nodes known)
- distribute compiled code over nodes acc. cost estimation
AMαχοS—AM for Web Querying

Are we alone in this?
- not quite: XLSTVM now part of Oracle DB
  - centralized query processing
  - very specialized instruction set for XLST 1.0/Oracle

algebra vs. abstract machine
- very similar idea: operators and their semantics
- but: usually tightly integrated (at least on physical layer)
- algebra for XML querying hot research issue

We have principles …
Enough vision already...

... on to the details ...
How to evaluate this?

- one “eval” operator?
- splitting it into base relations → conjunctive queries
  - root(v₀) ∧ child(v₀, v₁) ∧ label(v₁, “a”) ∧ child(v₁, v₂) ∧ label(v₂, “b”) …
  - much better for optimization (move “tough” decisions to compile-time)
  - but: naively done exponential
- compromise
  - path, twig operators: root(v₀) ∧ path(v₀, “a.b”, v₁) …
  - split at join and result variable
several variants but common principles:

basic type: node with properties

- element (structured) vs. content (atomic) nodes
- semi-structured data model with node identity
  - differs from previous Xcerpt DM (infinite regular trees)

memory model: memoization matrix

- non-1-normal-form table of operator results
- non-redundant (polynomial) store of query results
Three phase algorithm

- matrix **population**
  - evaluates only a spanning tree $T$ of operators from query $Q$
  - “directed” semi-joins $\rightarrow$ polynomial evaluation
- expansion of **non-tree joins** (similar to OO DBS case)
  - worst-case exponential in time and space
- matrix **consumption**
  - construction in the flavor of complex value algebra
Matrix population (spanning tree T of query Q)

- unary relations (property filters)
- binary & ternary relations (structural assembly)
  - basic relations (child, desc), (reg.) path operators, twig operators

Non-tree join expansion

- value, identity, and (direct) structural join

Matrix consumption

- basic constructors for each node type
  - grouping, aggregation, order, ...
Lot’s of **freedom at compilation**
- how to distribute operators between phase (1) and (2)
  - matrix population: semi-joins, but only acyclic CQ
  - join expansion: arbitrary shape, but exponential

“cover” areas for join variables to reduce exponent
- hypertree/query decomposition

**choosing the “right” operator**
- conjunction of base relations vs. twig operator

**supportive indices and DM variants**
- e.g., set-based vs. streaming (time vs. space)
## AMAXoS—Execution

The core of the core: the **evaluation** algorithm …

### Complexity

<table>
<thead>
<tr>
<th></th>
<th>Tree query</th>
<th>Graph query</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tree data</strong></td>
<td>$O(q \cdot v^2 + o)$</td>
<td>$O(v^q)$</td>
</tr>
<tr>
<td><strong>Graph data</strong></td>
<td>$O(q \cdot v \cdot e + o)$</td>
<td>$O(v^q)$</td>
</tr>
</tbody>
</table>

Table 1: Overview of Combined Time Complexity

$q$: number of query variables; $e$, $v$ number of edges, vertices resp., in the data; $o$: size of output
AM$\chi$oS—Execution

The core of the core: the \textbf{evaluation} algorithm …

\begin{figure}
\centering
\includegraphics[width=\textwidth]{chart.png}
\caption{Comparison of execution time with and without memoization for varying query sizes with fixed data size.}
\end{figure}

- Time (msec, logarithmic)
- Query size (variables)
- Without memoization
- With memoization

data size fixed
AMaxios—Execution

The core of the core: the **evaluation** algorithm …

query size fixed (~ 20 nodes)
### AM\(\chi\)oS—Architecture

And a way to **realize** them …

<table>
<thead>
<tr>
<th>Control Plane</th>
<th>Compilation API</th>
<th>Execution &amp; Answer API</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>simple observation and control API</td>
<td>control, observation, parameterization</td>
</tr>
<tr>
<td></td>
<td>compilation strategies</td>
<td>OO &amp; Web Service API</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program Plane</th>
<th>Parsing &amp; Validation Layer</th>
<th>Compilation Layer</th>
<th>Execution Layer (AM(\chi)oS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>program parsing and validation</td>
<td>unsatisfiable, tautological parts</td>
<td>pattern matching engine</td>
</tr>
<tr>
<td></td>
<td>multi-parser, normalization, modules</td>
<td>extensive query optimization</td>
<td>rule dispatcher and engine</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Plane</th>
<th>Schema Access Layer</th>
<th>Data Access Layer</th>
<th>Serialization Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>provides access to schema of data</td>
<td>incremental data access</td>
<td>incremental answer creation</td>
</tr>
<tr>
<td></td>
<td>type checking for compilation</td>
<td>storage and indexing engine</td>
<td>versatile Web format support</td>
</tr>
</tbody>
</table>
AMAxoS—Architecture

core layer: execution or AMAxoS proper ...

Abstract Machine AMAxoS

Rule Engine
Rule Dispatch
Pattern Matching Engine

Function Call

Static Function Library

Construction Engine

Substitution Sets

Answer Construction

 Memoization Matrix

Storage Manager

Storage & Index Hints

Code Segment
Hint Segment
Dependency Segment
AM Code

Rule 1
Rule 2
...

Query Compilation

Answer API

In-Memory Answer

Runtime Data Access Layer
AMαχοS—Architecture

The other core: **optimization** and compilation …

**Query Compilation**

**Logical Optimization**—Algebraic Optimization

- Translation logical algebra
  - patterns: annotated conjunctive queries over semi-structured graphs
  - rules: unfolding into complex value or object algebra where possible

- Rewriting system
  - elimination of dead and tautological query parts
  - join placement optimization
  - query compaction (common subexpressions)

**Physical Plan Generation**

**Query Classification**

determines class of query, e.g., to choose efficient alg. for sub-languages

**Operator Algorithm Selection**

determines realization of operators

**Index and Storage Model Selection**

selects in-memory representation and indices for data access

**Code Generation**

generate AM-code
  - direct representation of physical query plan
  - platform-independent
  - motion of invariant code
  - dead-code elimination

---

Typed AST
Translator
Canoninc
Logical Query Plan
Rewriting System
Optimized Logical QP
Physical Query Plan
Code Generator
AM Code
The end (of the talk) …

- novel approach to query execution
  - uniform platform for distributed evaluation
  - separation of querying and compilation

- lots of open issues, e.g.,
  - data structures
  - compilation & evaluation of high-level language constructs

1. “Compile once”
2. “Execute anywhere”
3. “Optimize all the time”