Aspects of Multiple Query Optimization for Data Analysis Applications

Henrique C.M. Andrade
(in conjunction with Tahsin Kurc, Alan Sussman, and Joel Saltz)
Department of Computer Science
University of Maryland, College Park, MD 20742
hcmma@cs.umd.edu
October 24, 2002

Definition: Multi-Query Optimization
- Goal: minimize the total cost of processing a series of queries by creating an optimized access plan for the entire sequence [Kang, Dietz, and Bhargava]
- My goal: minimize the total cost of processing a series of queries through data and computation reuse
  - The typical assumption in relational databases requires fine-grain temporal locality (due to the use of a global query plan), which is not really necessary!

Definition: Data Transformation
- Data and Computation Reuse
  - If we need an aggregate we have previously computed (and cached) we can substitute the request for computation by the result
  - Example: a=f(2)+f(1), will become a=9 by looking up the cache

- Data transformation
  - What if we don’t have the right “intermediate result”? 
  - Example: b=g(2)+g(3)
  - However we know that g(x)=f^2(x)+3!
  - A data transformation will compute g(2) and g(3) from f(2) and f(3)

Definition: Data Analysis Applications
- Processing Remotely-Sensed Data
  - NOAA Tiros-N w/ AVHRR sensor
  - AVHRR Level 1 Data
  - As the TIROS-N satellite orbits, the Advanced Very High Resolution Radiometer (AVHRR) sensor scans perpendicular to the satellite’s track.
  - At regular intervals along a scan line measurements are gathered to form an instantaneous field of view (IFOV).
  - Scan lines are aggregated into Level 1 data sets.
  - A single file of Global Area Coverage (GAC) data represents:
    - ~one full earth orbit.
    - ~110 minutes.
    - ~40 megabytes.
    - ~15,000 scan lines.
  - One scan line is 409 IFOV’s
Common Support for Data Analysis Applications

- Many data analysis applications have data processing similarities
- Especially true for applications dealing with range queries
- Ultimate Goal: middleware able to handle and optimize the execution of multiple query workloads transparently

Computational Model

- Memory
  - Shared Memory Configurations
  - Distributed Memory Configurations
  - Distributed Shared Memory Configurations [CCGrid'02]
- Computational Nodes
  - Homogeneous
  - Heterogeneous [SC'02]

System Architecture [SC'01, CCGrid'02, and SC'02]

- Query Server
  - Thread pool: one thread per query
  - Each query is represented by a customizable query object
  - Scheduler [IPDPS'02]
- Page Space Manager
  - Storage for input data (chunks and pages)
- Data Store Manager
  - Dynamic storage space of intermediate data structures and aggregates
  - Memory allocation and semantic lookups
  - Persistent Storage [LCR'02]

Outline of our Approach

- Semantic Cache [SC'01 & LCR'02]
  - Vanilla cache – hit or miss
  - Formalization of a data transformation model – partial hit
- Query Scheduling [IPDPS'02]
  - How the execution order affects performance
- Cache Replacement Policies [WC3'02]
  - How the replacement policy affects performance
- Functional Decomposition [CS-TR-4404]
  - Automatically extracting maximum data and computation reuse
Active Semantic Cache - Data Transformation Model

- Query predicates $M_i$ and $M_j$ store bounding box and zoom factor

\[
\text{overlap index} = \frac{A_i}{A_j} \times \frac{A_j}{A_i}
\]

Performance Results

Using 4 processors (up to 4 queries simultaneously)

<table>
<thead>
<tr>
<th>PDSS size</th>
<th>128M</th>
<th>192M</th>
<th>256M</th>
<th>320M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
</tr>
</tbody>
</table>

Disabled | Basic | Projection Aware

Query Scheduling

- Given the query execution framework and the data transformation model, if we have many queries sitting in the waiting queue, in which order should we go about them?

Scheduling Policies

- Most Useful First (MUF)
  - Tries to schedule queries that are going to be the most beneficial to other queries earlier
- Farthest First (FF)
  - Schedules queries that are "far" apart to avoid interlocking and "idling"
- Closest First (CF)
  - Schedules queries that are "close" to improve locality
- Closest and Non-Blocking First (CNBF)
  - Schedules queries that are "close" to improve locality, but tries to avoid interlocks
- Baseline cases: FIFO (First-In, First-Out), SJF (Shortest Job First)
Scheduling Policies

<table>
<thead>
<tr>
<th># of threads</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>16</td>
<td>50</td>
</tr>
</tbody>
</table>

Observations
i) FIFO is much worse as expected.
ii) FF, CF, and CNBF beat SJF – more reuse.
iii) Performance increases up to 4 simultaneous queries.

Cache Replacement Policies

- Query Scheduling and Cache Replacement Policies are complementary
  - Scheduling tries to order execution to better use what’s in the cache
  - Replacement policy tries to maintain the working set around
- Twist: not your run-of-the-mill replacement policy
  - difference in costs for building an intermediate aggregate
  - Example: An aggregate for application A takes 100 seconds to be built, another for application B takes 10 seconds. Both are equal candidates for eviction, which one should be evicted?

Cache Replacement Policies

- Least Recently Used (LRU)
  - Replaces the aggregate that has been requested least recently
- Size
  - Replaces the largest aggregates
- Least Frequently Used (LFU)
  - Evicts the aggregate used less frequently
- Least Relative Value (LRV)
  - Replaces the aggregate with the smallest value
  - Issue: which metrics to use for value

Overall Workload Execution Time (FIFO scheduling)

Overall Workload Execution Time (SJF scheduling)

Configuration:
- 16 clients (8 querying image 1, 6 querying image 2, and 2 querying image 3)
- 8 clients using Averaging and 8 using Subsampling
- 32 queries per client

Observations:
1. LRVB and ALRVB outperform LRU by as much as 8% (FIFO), by as much as 16% (SJF).
2. Aging is effective
3. LFU gets relatively more for SJF scheduling.
4. When cache is big enough for the working set, policies tend to behave similarly.
Functional Decomposition

- Reuse is a powerful mechanism for improving performance
- Exposing reuse sites
- Framework for automatically employing data and computation reuse

Case Study - Kronos

- Remote sensing
  - AVHRR datasets
  - 5-spectral bands
  - 1GB per day
- Visualization
  - Multiple correction and compositing algorithms
  - Cartographic projections
- Use: crop yield prediction, wild fire prediction, etc.

Types of Reuse for Data Analysis Applications

- Dimensional (Spatio-Temporal) Overlap (DO)
- Composable Reduction Operations (CR)
- Invertible Functions (InvF)
- Inductive Functions (IndF)

Dimensional Overlap
Composable Reduction Overlap

Day 1
Day 2
Day 3
Composite (MAXNDVI) Day 1-7

Inductive Aggregation

cached
projected
query result
merged
automatically computed subquery

Query Execution Process

- Locating reuse site
  - Strategy
  - Congruence
- Employing a reuse strategy
  - Project function templates

Experimental Evaluation

- Workload Model
  - Customer Behavior Model
  - Graph (CBMG)
  - How a real user interacts with Kronos

<table>
<thead>
<tr>
<th>Transition</th>
<th>Workload I</th>
<th>Workload II</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Point of Interest</td>
<td>5%</td>
<td>2%</td>
</tr>
<tr>
<td>Spatial Movement</td>
<td>10%</td>
<td>40%</td>
</tr>
<tr>
<td>New Resolution</td>
<td>5%</td>
<td>40%</td>
</tr>
<tr>
<td>Temporal Movement</td>
<td>5%</td>
<td>5%</td>
</tr>
<tr>
<td>New Correction</td>
<td>10%</td>
<td>20%</td>
</tr>
<tr>
<td>New Compositing</td>
<td>5%</td>
<td>20%</td>
</tr>
<tr>
<td>New Compositing Level</td>
<td>50%</td>
<td>10%</td>
</tr>
</tbody>
</table>
**Is Functional Decomposition effective?**

**Workload 1**

<table>
<thead>
<tr>
<th>Orig Kronos</th>
<th>All</th>
<th>Selection</th>
<th>Correction</th>
<th>Compositing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25.60</td>
<td>31.25</td>
<td>28.07</td>
<td>34.48</td>
</tr>
<tr>
<td>224.37</td>
<td>153.83</td>
<td>181.58</td>
<td>166.77</td>
<td>206.28</td>
</tr>
</tbody>
</table>

**Average Time per Query (s)**

**Workload 2**

<table>
<thead>
<tr>
<th>Orig Kronos</th>
<th>All</th>
<th>Selection</th>
<th>Correction</th>
<th>Compositing</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>35.37</td>
<td>43.39</td>
<td>37.26</td>
<td>56.55</td>
</tr>
<tr>
<td>316.31</td>
<td>200.76</td>
<td>252.61</td>
<td>217.51</td>
<td>337.44</td>
</tr>
</tbody>
</table>

**Average Time per Query (s)**

**Configuration**
- 16 clients
- 4 queries each
- Client only submits queries when the last one has been computed

**Projection Functions**

**Combinations of Projection Functions**

**Configuration**
- 32 queries using workload 1

**<That's all Folks!>**

- Summary
- Conclusion