Type 2 Slowly Changing Dimensions:

A Case Study Using the Co>Operating System

Craig Stanfill
Ab Initio Software
DOLAP’12, Maui
Overview

The Co>Operating System

- Ab Initio’s parallel computing framework
- Based on partitioned dataflow
- Graphic programming

A little about what graphs really look like

- Primary computation
- Secondary computations: “Salad Dressing”
- Five solutions to “Type 2 Slowly Changing Dimensions”

Performance:

- Scalability
- Insights into the important tradeoffs for optimization
The Co>Operating System

Parallel framework for Enterprise Computing
- Widely used for ETL, Data Warehousing
- Widely used for Mission Critical Realtime Apps
- Stock Exchanges, Telecommunications, Credit Card Processing
- Batch, Streaming, Service, Transactional Modes

Compared with MapReduce:
- Broader applicability
- More built-in functionality
- Handles extreme levels of complexity
Parallel Graphic Dataflow

Real World: Deal with Errors, Log Output, Auditing Etc
Graphs Nest Very Deeply

9 Levels Deep; 33 Subgraphs; 259 Components
The Problem: Type 2 Slowly Changing Dimension

- Raw Dimension
- Cooked Dimension
- Old Surrogate Key Map
- New Surrogate Key Map
- Other Annoying Stuff
- Join + Salad Dressing
3 Cases

<table>
<thead>
<tr>
<th>Case</th>
<th>Raw Dimension</th>
<th>Old Keymap</th>
<th>Cooked Dimension</th>
<th>New Keymap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>Huge</td>
<td>Empty</td>
<td>Huge</td>
<td>Big</td>
</tr>
<tr>
<td>Full Reload</td>
<td>Huge</td>
<td>Big</td>
<td>Small</td>
<td>Big</td>
</tr>
<tr>
<td>Incremental Reload</td>
<td>Small</td>
<td>Big</td>
<td>Small</td>
<td>Big</td>
</tr>
</tbody>
</table>

Which cases do you optimize for?

- Initial Load: Big Job, Only Once
- Full Reload: Room for optimization
- Incremental: Lots of room for optimization
  (May not be a viable option)
Solution 1: Sort Merge Join

Depth: 2
Subgraphs: 3 (all shared)
Components: 27
Salad Dressing: Handle Dups and Rejects

This is a **reusable subgraph**
Salad Dressing: Audit and Update Surrogate Key Seed

Read paper for how we generate surrogate keys
Solution 2: Hybrid Hash Join
Solution 3: Lookup Files

Diagram showing the process flow for Lookup Files:
- Raw Dimension
- Partition by Key
- Scan: Map Records and Assign Surrogate Keys
- Categorize Records
- Handle Rejects and Duplicates
- Audit and Update Surrogate Key Seed
- Update Surrogate Key Lookups
- Processed Subscribers
- Lookups
Solution 3: Lookup File Optimization
Solution 4: Keep Surrogate Keys in Database
Solution 5: Stored Procedure
Wall Clock Time (32 ways parallel)

- **Load**
  - MJ
  - HJ
  - LU
  - SQL
  - SP

- **Full Update**
  - MJ
  - HJ
  - LU
  - SQL
  - SP

- **Incremental**
  - MJ
  - HJ
  - LU
  - SQL
  - SP
Cores Used: MPP (Lookup Solution, Initial Load)

1 Node

4 Nodes
CPU Time/Record (Lookup Solution, Initial Load)

1 Node

4 Nodes
Pipeline Factor

MJ  HJ  LU  SQL  SP

1  2  4  8  16  32
Questions?