Slalom: Coasting Through Raw Data via Adaptive Partitioning and Indexing

Presented by Ketill Guðmundsson
The problem

• Data generated has grown massively
  • Sensor data
  • Network monitoring data
  • etc

• Current analytical system not built for this much data
  • Data loading is expensive
  • Time-to-insight rising

• Analytical queries are hard to predict
  • No or little workload knowledge
Current solution

• Working over raw data
  • No data loading cost
  • Only logical indexes
  • Physical data is never changed

• In-situ queries
Current solution problems

• Using a indexed DBMS overtakes in-situ in time
  • High initial cost
  • Low query cost
• Ideally that never happens
Slalom

• Dynamic partitions
  • Logical partition only
  • Created at runtime

• Dynamic indexes
  • Bloom filters
  • Zonemaps
  • Bitmaps
  • B+ Trees
Slalom architecture
## Slalom statistics

<table>
<thead>
<tr>
<th>Data (partition $i$)</th>
<th>Data (global)</th>
<th>Queries (partition $i$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_i$: mean value</td>
<td>$Size_{page}$: page size</td>
<td>$C_{build}$: index build cost</td>
</tr>
<tr>
<td>$min_i$: min value</td>
<td>$Size_{file}$: file size</td>
<td>$C_{fullscan}$: full scan cost</td>
</tr>
<tr>
<td>$max_i$: max value</td>
<td></td>
<td>$LA_i$: #q since last access</td>
</tr>
<tr>
<td>$dev_i$: std. deviation</td>
<td></td>
<td>$AF_i$: part. access freq.</td>
</tr>
<tr>
<td>$DV_i$: #distinct values</td>
<td></td>
<td>$sel_i$: avg. sel. (0.0-1.0)</td>
</tr>
</tbody>
</table>
Partition manager

- Only logical partitions
- Contiguous and non-overlapping
- Iterative refinement
Partition manager
Partition manager

- Incremental splits
- Stops when partition is stable
- Splits into many smaller partitions

\[ m = \frac{N \cdot (sel + \log_b (1 - sel))}{\log_b \left( \frac{\sqrt{2 \cdot \pi \cdot sel \cdot N}}{2} \right)} \quad \text{where} \quad b = \frac{e}{sel \cdot (1 - sel)} \]
Index manager

• Only applied to stable partitions
• Value existence
  • Bloom filters
  • Zone maps
• Value position
  • B+ Tree
Index manager – Which index?

i. the cost of building the index, which corresponds to the case where the building of the index will take place at time $i$. Index construction takes place as a by-product of query execution and includes the cost of the current query.

ii. the cost of using the index, which corresponds to the case where the index has already been built.

iii. the cost of queries doing full partition scan, which corresponds to the case for which the index will not be built.

$$E = \sum_{i=1}^{T} \left( p_i \cdot C_{\text{build}, idx} + \sum_{j=1}^{i-1} p_j \cdot C_{\text{use}, idx} + (1 - \sum_{j=1}^{i-1} p_j) \cdot C_{\text{use}, fs} \right)$$
Experiments

Figure 4: Sequence of 100 queries. Slalom dynamically refines its indexes to reach the performance of an index over loaded data.

Figure 5: A breakdown of the operations taking place for Slalom during the execution of a subset of the 1000 point query sequence.
Experiments

**Figure 6:** Sequence of 1000 queries. Slalom does not incur loading cost and dynamically builds indexes.

**Figure 7:** Memory consumption of Slalom vs. a single fully-built B+ Tree for PostgreSQL and DBMS-X. Slalom uses less memory because its indexes only target specific areas of a raw file.
Experiments

**Figure 10**: Slalom performance using different memory budgets. Slalom performance varies with allotted memory.
Experiments

Figure 12: Cracking techniques converge more efficiently but Slalom takes advantage of data distribution.
Conclusion – Positive points
Conclusion – Negative points

Figure 12: Cracking techniques converge more efficiently but Slalom takes advantage of data distribution.