Efficient Data Management and Statistics with Zero-Copy Integration

Jonathan Lajus & Hannes Mühleisen

SSDBM 2014, 2014-06-30
Collect data

Load data

Filter, transform & aggregate data

Analyze & Plot

Data Management System

Statistical Toolkit

Bottleneck, thanks David!

Publish paper/Profit
Data Transfer Options

- “Socket-Style”, JDBC, ODBC, DBI, …
  - Serialization, copy, copy, copy, copy, Deserialization
- In-process embedding
  - No sockets (hopefully), but still conversion
- Shared memory
- No transfer altogether by extending DB or stats
Zero-Copy

• In the end, C-arrays of native types are everywhere
  • Hardware does not like Java objects (usually)
  • Hmm…

• In-process data sharing by passing pointers
  • If both systems are based on C arrays, we could get away with metadata management
(1) Statistics → Database

(2) Statistics → Database → Query Result

(3) Statistics → Database ← Query Result
Challenges

• Compilation & Linking
  • symbol name clashes are likely

• Read/Write synchronization

• Memory management (who calls free())

• NA/NULL value encoding?

• Complex Objects
Proof-of-Concept

Analyze & Plot

Filter, transform & aggregate

https://github.com/lajus/monetinr
R SEXP

MonetDB BAT
Dress-up

+ Garbage Collection Fun
Experiments

• data.table, high-performance R data access
• MonetDB.R, DBI/socket-based DB access
  • Equivalent systems, different connection
• RSQLite, embedded SQL database
  • Still needs conversion
Predictions

• Dedicated data management systems should be better at data management than pimped stats tools

• In-process integration should make a big difference once result sets get large

• Column store performance gain should be visible
Setup

• Typical data management tasks
  • Selection & Projection
  • Aggregation
  • Joins

• 10MB, 100MB, 1GB, 10GB datasets

• Desktop-class machine, 16 GB RAM, 3.4 Ghz i7, Fedora Linux
Beats Sockets & Stats Extensions

Execution Time (log)

Dataset Size (log)

- 1% Rows Selected
- 10% Rows Selected
- 50% Rows Selected

Data points for different dataset sizes (10 MB, 100 MB, 1 GB, 10 GB) and execution times (10ms, 100ms, 1s, 10s, 1min, 10min) are illustrated. The graph compares various database technologies, including data.table, RSQLite, MonetDB.R, and a Prototype, indicating performance improvements over Sockets & Stats Extensions.
Complex queries / small result sets not worth it
Conclusions

• Zero-Copy possible

• Vast performance benefits

• But
  • Read/Write access?
  • Iterative processes?
  • Optimization?
Special Thanks

• Thomas Lumley (R)

• Sjoerd Mullender (MonetDB)
R UDFs in MonetDB

CREATE FUNCTION kmeans (data FLOAT, ncluster INTEGER) RETURNS INTEGER LANGUAGE R {
  kmeans(data,ncluster)$cluster
};

Watch the next MonetDB release…
quantile(c(.05,.95))

PL/R

R in MonetDB

<table>
<thead>
<tr>
<th>sys</th>
<th>sqltime</th>
<th>dumbtime</th>
<th>udfime</th>
<th>vdumbtime</th>
<th>plrtime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time (s)

Rows

1 K 10 K 100 K 200 K 1 M 10 M 100 M
Thank You!

Questions?