Invisible Glue: Scalable Self-Tuning Multi-Stores

Francesca Bugiotti§, Damian Bursztyn§, Alin Deutsch¶, Ioana Ileana§, Ioana Manolescu§

§ OAK team, INRIA, France
¶ DB group, UC. California San Diego
The problem

- **Glut of varied** data management systems (DMS)
  - DM includes DBMS
- **Different data models:**
  - Relational, nested relational, tree, k-v, graphs, ...
- Different **data access capabilities** (from simple API to various query languages)
- Different **architectures:** disk- vs. memory-based, centralized vs. distributed etc.
- Different **performance**
- Different levels of **transaction support**
The problem

- **Glut of varied** data management systems (DMS)
  - DM includes DBMS
- Different **data models**:
  - Relational, nested relational, tree, k-v, graphs, ...
- Different data access capabilities (from simple API to various query languages)
- Different architectures: disk vs. memory-based, centralized vs. distributed, etc.
- Different **performance**
- Different levels of **transaction support**
The problem

• **Glut of varied** data management systems (DMS)

How do we get **performance** for a variety of datasets on a variety of architectures?

- Different data models:
  - Relational,
  - Nested relational,
  - Tree,
  - K-v,
  - Graphs,
  ...

- Different data access capabilities (from simple API to various query languages)

- Different architectures:
  - Disk vs. memory-based,
  - Centralized vs. distributed etc.

- Different performance

- Different levels of transaction support

Focus **not on beating the most specialized optimizations** of the most specialized engine for a **given model/application**.

Bugiotti, Bursztyn, Deutsch, Ileana, Manolescu / Estocada / CIDR 2015
The problem

- **Glut** of varied data management systems (DMS)
- Different data models: Relational, nested relational, tree, k-v, graphs, …
- Different data access capabilities (from simple API to various query languages)
- Different architectures: disk vs. memory-based, centralized vs. distributed, etc.
- Different performance
- Different levels of transaction support

How do we get **performance** for a variety of datasets on a variety of DMS?

Focus not on **beating the most specialized optimizations** of the most specialized engine for a given model/application.

Focus on **robust performance for varied data models** across a changing set of heterogeneous DMSs.

Bugiotti, Bugiotti, Deutsch, Ileana, Manolescu / Estocada / CIDR 2015

NoSQL DMSs
The problem, qualified

- Glut of varied data management systems (DMS)
  - DM includes DBMS

- Different data models:
  - Relational, nested relational, tree, k-v, graphs, ...

- Different data access capabilities (from simple API to various query languages)

- Different architectures:
  - Disk vs. memory-based, centralized vs. distributed etc.

- Different performance

How do we get performance for a variety of datasets on a variety of DMSs

Automatically

With correctness guarantees

With no hassle for the application layer

Resilient to changes
A piece of self-derision: The Next Data Model That Will Save The World

New Data Model: early days
First Papers:
- First formalism (relatively simple and clean)
- First query language, toy implementation

New Data Model: golden days
For Previous Data Model, Everything Is Undecidable!

New Application: Previous Data Model Can’t Do It!

This Simple Model Needs Extensions!
The Baseline Method Needs Optimizations!
Self-Tuning Technique for New Data Model

New Data Model: Hybrid solutions
You Can Have This Data Model and the Previous One!

New Data Model: Standardization (?)
Blessing/curse: Industrial adoption

Formerly New Data Model: we’re done with it
“XML Research Is Passé”

Bugiotti, Bursztyn, Deutsch, Ileana, Manolescu / Estocada / CIDR 2015
Estocada: invisible glue for heterogeneous stores

- Data models: \textit{any/many} (side by side)
  - As the data is
- Systems: \textit{any/many} (side by side)
  - Those available
- Store each data set as a set of \textit{fragments}
  - Or splits / shards / partitions / indexes / materialized (potentially indexed)
  - Each fragment resides in a DMS

Bugiotti, Bursztyn, Deutsch, Ileana, Manolescu / Estocada / CIDR 2015
## Dataset fragmentations

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccc}
A & B & C & D \\
1 &   &   &   \\
2 &   &   &   \\
3 &   &   &   \\
4 &   &   &   \\
5 &   &   &   \\
6 &   &   &   \\
\end{array}
\]

\[
\begin{array}{c}
A \\
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
\end{array}
\]

\[
\begin{array}{c}
A \\
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
\end{array}
\]

\[
\begin{array}{c}
A \\
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
\end{array}
\]

\[
\begin{array}{c}
A \\
5 \\
6 \\
\end{array}
\]

\[
\begin{array}{c}
A \\
5 \\
6 \\
\end{array}
\]

---

Bugiotti, Bursztyn, Deutsch, Ileana, Manolescu / Estocada / CIDR 2015
Dataset fragmentations

Example: relational dataset R
Dataset fragmentations

Example: relational dataset R
Dataset fragmentations

Example: relational dataset R
Dataset fragmentations

Example: relational dataset R
Dataset fragmentations

Example: relational dataset R
Dataset fragmentations

Example: relational dataset R
Dataset fragmentations

Example: relational dataset R
Fragmentations made of views

• The content of each fragment is described declaratively

• **Fragment** = *(materialized) view* [+ parameters]
  - « The name and addresses of all clients »
  - « The sales partitioned by zipcode »

• **Index** = view with binding pattern
  - « The name and addresses of all clients, by their age and zipcode »
  - Also: navigation in trees or graphs key-value stores

**Fragment** = materialized view [+ parameters] [+ binding pattern]
Fragments distribution across stores
Fragments distribution across stores
Fragments distribution across stores

Bugiotti, Bursztyn, Deutsch, Ileana, Manolescu / Estocada / CIDR 2015
Fragments distribution across stores

Bugiotti, Bursztyn, Deutsch, Ileana, Manolescu / Estocada / CIDR 2015
Fragments distribution across stores
Fragments distribution across stores

Data model translation
applied at loading
• « Dumb » translation
• The extraction logic is in the view

Bugiotti, Bursztyn, Deutsch, Ileana, Manolescu / Estocada / CIDR 2015
Fragments distribution across stores

Applications query the data in native format
Fragments distribution across stores

Fragment description by views guarantees properties such as: completeness, inclusion, equivalence, etc.
Query answering = **View-Based Rewriting**

- **VBR** known for dramatic performance improvements
  - Basically no limit (e.g. view = query)!
  - P. Larson (2011 SIGMOD ToT): « the problem is explaining to a disappointed user that a modified view doesn’t match any more! »

- **Comparison with « Local As Views » mediation**
  - ≠ data models

Common data model (V1, ..., Vn, Q)  

Native dataset model

---

Bugiotti, Bursztyn, Deutsch, Ileana, Manolescu / Estocada / CIDR 2015
Query answering = view-based rewriting

- VBR known for dramatic performance improvements
- Comparison with « Local As Views » mediation:
  - ≠ data models
  - Side-by-side data models at the top

→ Common benefit with LAV: Applications unaware of the fragmentation!
→ Novel benefit: fragments can migrate to ≠ systems and data models
View-based rewriting with heterogeneous data models

• Model all problems as the most complex and solve it there
  – No! (complexity; also: The Next Data Model that will Save the World... 😊)

• Have a set of side-by-side rewriting algorithms (dataset model M0, DMS models M1, M2, ...)
  – Impractical due to many data model combinations

• Our solution
  – Identify a small set of « core » data models which include the others
  – Have side-by-side rewriting algorithms for this small core (hint: many reuse opportunities)
VBR with heterogeneous data models

• At the core:
  – The capacity to describe dataset properties
    • **Structure**: (nested) tuples, trees, graphs, bags
    • **Constraints**: keys, foreign keys, inclusion dependencies (e.g. also RDF semantics)
  – A rewriting algorithm capable of leveraging all the information to find all equivalent query rewritings
    • Constraints enable rewritings in cases when there would be none without them!
    • Starting point: efficient Chase-and-Backchase algorithm [Ileana, Cautis, Deutsch, Katsis, SIGMOD 2014]
• VBR
  – Outputs: queries to DMSs (in their native language) + remaining integration operations
  – DMS capability descriptions exploited here.

• Runtime
  – To perform integration operations
  – For this, a single runtime (for the most expressive model, e.g. nested relations), should do
  – We may borrow one of the DMS’s runtime
What about performance?

- Select the rewriting likely to lead to the best query evaluation performance
  - **Cross-system cost model**
    - Not as crazy as it looks (cost model calibration ’86; it works!)
    - Modest extension for binding patterns

- View recommendation
  - « Cross-model, cross-system data storage advisor »
  - Great progress in recent years on single-model storage (view, index etc.) recommendation
  - **Combinatorial problem** (select a subset of the possible views minimizing cost estimation)
Closest related work

• Mixed-model VBR
  – Agora [Manolescu, Florescu, Kossmann, VLDB 2001], Mars [Deutsch & Tannen, VLDB 2003]: XML

• Running a DMS on top of heterogeneous stores
  – Federated databases (Tomasic, Valduriez et al., 1996)
  – Data integration (wrappers / mediators) up to VIDA [Karpathiotakis, Alagiannis, Heinis, Branco, Ailamaki, CIDR 2015]
  – Recent hybrid systems [LeFevre, Sankaranarayanan, Hacigumus, Tatemura, Polyzotis, Carey, SIGMOD 2014]; [Jindal, Quiané-Ruiz, Dittrich, CIDR 2013]
Related issues

• Orthogonal concerns
  – Offering users a single integrated view
    • A la « global-as-views » data integration
  – Cleaning and extracting the data

• Not so orthogonal concerns
  – Providing common transactional properties across different DMS
  – « Failsafe mode »: use Estocada only for performance on top of a « vanilla » store
  – « Gradual »: failsafe ➔ monitor application ➔ recommend fragment deployment)
Sample application: smart city data integration

- Datalyse French R&D project
  - **Relational** transport database
  - **RDF** Open Data produced by the city administration (cultural artefacts, events...)
  - **Graph** social network data harvested from various applications
  - May be used with or without **log** data from city Web site and various apps

- Mid-size IT companies clueless on what to use
- Easy to be wrong by orders of magnitude
Wrap-up

• **Goal:**
  – Handle data efficiently on top of a given set of systems, even if it changes
  – Make the most out of each system

• **Method:**
  – View-based rewriting for multiple data models, under constraints

• **Status:** VBR ongoing, recommendation next

Thank you / Questions?