Distributed Queries without Distributed State

Vassilis Papadimos  David Maier
{vpapad,maier}@cse.ogi.edu

OGI School of Science & Engineering
Oregon Health & Science University
Distributed Data vs. Distributed Queries

The Internet is arguably the most successful distributed system ever. We have:

- A flexible representation for data (XML)
- Soon, a query language (XQuery)
- An abundance of distributed data
- No easy way to ask ad-hoc distributed queries
Distributed Query Processing

- Optimization
- Deployment
- Execution
Distributed Query Processing

- Optimization
- Deployment
- Execution
Distributed Query Processing

- Optimization
- Deployment
- Execution
Distributed Query Processing

- Optimization
- Deployment
- Execution
Distributed Query Processing

- Optimization
- Deployment
- Execution
... doesn’t scale well!

- Distributed execution, *centralized optimization*
- Coordinator must have *accurate, detailed* knowledge:
  - data placement and statistics
  - server capabilities and load
  - network conditions
- All sites must be up and willing to work on the query throughout the deployment and execution phases
Distributed State Considered Harmful

• Partial evaluation: allow servers to evaluate as much of a query as they want to

• Distributed optimization: allow servers to optimize their work independently

• Decoupled execution: minimize the time servers have to wait for each other, even if that means abandoning pipelined execution
Mutant Query Plans (MQPs)

- Query plan graphs, serialized in XML, that may contain, apart from regular query operators:
  - Abstract resource names (URNs)
  - Pointers to concrete resource locations (URLs)
  - Verbatim XML data

- A server can mutate a query plan by:
  - Resolving URNs to URLs, or URLs to XML data
  - Evaluating subplans, inserting results in their place

- Server then routes the MQP to the “next” server
• Catalog: maps URNs to URLs, or to other servers that can resolve them
A classic example

“Find every line of Sir John Falstaff, in all the plays of Shakespeare”

\[
\text{display address=129.95.50.105:9020}
\]

\[
\text{select speaker="Falstaff"}
\]

\[
\text{unnest line,speaker}
\]

\[
\text{urn:niagara:shakespeare}
\]
A classic example

More specific URNs

```
display address=129.95.50.105:9020

select speaker="Falstaff"

unnest line,speaker

union

urn:niagara:shakespeare.comedies
urn:niagara:shakespeare.tragedies
```
A classic example

Pushing operators through union

display address=129.95.50.105:9020

union

select speaker="Falstaff"

unnest line,speaker

urn:niagara:shakespeare.comedies

select speaker="Falstaff"

unnest line,speaker

urn:niagara:shakespeare.tragedies
A classic example

Resolving a URN to one or more URLs

```sql
display address=129.95.50.105:9020
union
select speaker="Falstaff"
unnest line,speaker
http://...
http://...
http://...
select speaker="Falstaff"
unnest line,speaker
urn:niagara:
shakespeare.tragedies
```
A classic example

Evaluating a sub-plan

display address=129.95.50.195:9020

union

select speaker="Falstaff"

unnest line,speaker

urn:niagara:shakespeare.tragedies
A classic example

Finally, reducing the plan to a constant piece of XML, and shipping it to the client

display<sub>address=129.95.50.195:9020</sub>

<line>
  Now, Master Shallow, you’ll complain of me to the king?
</line>
<line>
  But not kissed your keeper’s daughter?
</line>
...
Performance

- Longer latency
  - Only one server works at the query at a time

- Smaller *footprint* (total time spent in participating servers)
  - Never have to wait for other servers

- Some preliminary results
Mutant Query Optimization

What are we optimizing for?

- Time until client gets (first/all) results
- Total resources used
- Resources used in this server
- Resulting MQP size
  - Materializing local results in space, not in time
Consolidation

- Move locally evaluable operators together

- Rules for consolidating any plan with joins, unary operators and scans

- Full consolidation not always possible
Consolidation

- Move locally evaluable operators together
- Rules for consolidating any plan with joins, unary operators and scans
- Full consolidation not always possible
Absorption

Transformations that allow us to do some work locally, even on plans that cannot be consolidated further
Deferment

Refrain from evaluating a locally evaluable sub-plan to avoid inflating the result size

What if $B \Join (C \Join A)$ is a cartesian product?
Related Work

- Parachute Queries (Bonnet and Tomasic)
  - Partial evaluation of queries with unavailable sources

- ObjectGlobe (Braumandl et al.)
  - Centralized metadata maintenance, query optimization

- Intensional Answers (Jim and Suciu)

- ubQL (Sahuguet et al.)
  - MQPs similar to “recruiting” strategy

- P2P (a cast of thousands!)
Future Work

• Working on implementing consolidation, deferment and absorption strategies in our prototype using Columbia
  – Cost estimation is hard

• Looking for automated ways to propagate structured metadata among MQP servers
  – Piggybacking metadata updates on queries?
Conclusions

We believe that MQPs provide a viable, scalable framework for Internet–wide distributed queries

- No need for 100% accurate, centralized metadata
- No separate deployment phase
- No need for centrally–coordinated execution
- Optimize as you go!
Performance

• Trading away pipelined execution and low latency for scalability and throughput

• Comparing our pipelined and MQP prototypes on the toy “Falstaff” query

• Server $B$ stores comedies, $C$ histories and tragedies
  
  – Pipelined version: $A$ is the coordinator, $B$ and $C$ forward Falstaff lines to $A$
  
  – MQP version: $A$ resolves URNs to $B$ and $C$, $B$ inserts comedy lines and forwards to $C$, $C$ appends tragedy lines and returns to the client
Results

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Normal load</th>
<th></th>
<th>Heavy load on $C$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Latency</td>
<td>Footprint</td>
<td>Latency</td>
<td>Footprint</td>
</tr>
<tr>
<td>Pipelined</td>
<td>11.0s</td>
<td>27.3s</td>
<td>19.1s</td>
<td>43.6s</td>
</tr>
<tr>
<td>Mutant</td>
<td>13.9s</td>
<td>14.6s</td>
<td>26.1s</td>
<td>28.0s</td>
</tr>
</tbody>
</table>
We can consolidate any expression that contains just joins and scans using repeated applications of these rules:

\[
R \Join L \rightarrow L \Join R
\]

\[
L_1 \Join (L_2 \Join R) \rightarrow (L_1 \Join L_2) \Join R
\]

\[
R_1 \Join (L \Join R_2) \rightarrow (L \Join R_1) \Join R_2
\]

\[
L \Join (R_1 \Join R_2) \rightarrow (L \Join R_1) \Join R_2
\]

\[
R_1 \Join (R_2 \Join R_3) \rightarrow (R_1 \Join R_2) \Join R_3
\]