Holistic Application Analysis for Update-Independence

Charlie Garrod
with
Amit Manjhi, Bruce M. Maggs, Todd Mowry, Anthony Tomasic

School of Computer Science
Carnegie Mellon University
Scaling Web applications with database caches

- Proxy servers mimic 3-tier architecture
  - Use DB cache at proxy nodes
Scaling Web applications with database caches

- Proxy servers mimic 3-tier architecture
  - Use DB cache at proxy nodes

Problem: On update U, where to send U?
One solution: publish / subscribe for cache consistency management

• On caching a query Q:
  – Subscribe to messages for updates that affect Q

• On an update U:
  – Publish U to notify all affected query caches

The challenge: Relate (current) query to (future) possible updates that affect it
Our previous work: Analyze the Web application’s DB requests

- Determine which updates affect each query
  - Create subscription for any query-update pairs which cannot be ruled independent
- Query and update templates
  - e.g. SELECT name FROM emp WHERE salary > ?
  - UPDATE emp SET dept = ? WHERE id = ?
  - value set at run-time
This talk: *Holistic analysis*

- Compiler-like analysis of how the Web application uses the DB data
  
  - e.g.: `emp:`

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>dept</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
try {
    ResultSet r = connection.executeQuery(
        "SELECT * FROM emp WHERE salary > 40");
    while (r.next())
        System.out.println(r.getString("name"));
}
```

*Update:* 

```
UPDATE emp SET dept = ?
WHERE id = ?
```
Talk outline

• The DB cache update problem and holistic application analysis
  → Implementing holistic analysis
    – Existing-primary-key optimization
• Evaluating holistic analysis
• Related work and closing remarks
Implementing holistic analysis

1. Associate application data with DB data where possible
2. Apply basic control-flow and data-flow compiler algorithms
3. Use data dependence analysis to prove application not affected by updates
1. Associating app and DB data

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>dept</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Simple table/column associations only
- Rely on statically-declared access into the ResultSet

```java
try {
    ResultSet r = connection.executeQuery("SELECT * FROM emp WHERE salary > 40");
    while (r.next())
        String rName = r.getString("name");
        int rSalary = r.getInt(4);
        rName: emp.name
        rSalary: emp.salary
    }
```
2. Compiler-like analysis

- DB-app annotations are static compile-time “constants”
- Propagate annotations to where the data is used
  - Break program into basic blocks
  - Propagate annotations as constants
3. Data dependence analysis

• [Blakeley+’89, Elkan’90]
  – If an update does not write any data read by a query, then the update cannot affect the query

• [Olston+’05, Garrod+’08]
  – Extended to DB templates

• Extending to apps: data-use analysis
  – Can take advantage of propagated annotations
Two optimizations we implemented:

1. *Unused-data* optimization
   - Application does not use affected DB data

2. *Existing-primary-key* optimization
   - Constrain run-time template parameters
   - Application does not execute the template instantiations affected by an update
Constraining parameter values

```
try {
    int x = 4;
    PreparedStatement t = connection.prepareStatement(
        "SELECT dept FROM emp WHERE id = ?");
    t.setInt(1, x);
    ResultSet r = t.executeQuery();
    ...
}
```

**Update:**
```
UPDATE emp SET dept = ?
WHERE id = 2
```
Constraining parameter values

1: $X \leftarrow \text{result of Query 1}$
2: $X$ used as parameter in Query 2

• Known constraints from Query 1 may aid analysis for Query 2
A quiz for you:

Does this update template ever affect a result of the query template?

**Query:** SELECT * FROM emp WHERE id = ?

**Update:** INSERT INTO emp
VALUES(NULL, ?, ?, ?)

**Answer:** Yes, but only for a query on a non-existent id

- id is emp’s primary key
- id is auto-assigned to be unique
Existing-primary-key constraint

- id is emp’s primary key

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>dept</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Query 1: SELECT id FROM emp WHERE name = ?
Query 2: SELECT * FROM emp WHERE id = ?
Update: INSERT INTO emp
VALUES( NULL, ?, ?, ?, ?)

- id is auto-assigned to be unique

• If Query 2 always uses id from Query 1, then Query 2 is unaffected by Update
Talk outline

- The DB cache update problem and holistic application analysis
- Implementing holistic analysis
  - Unused-data optimization
  - Existing-primary-key optimization

→ Evaluating holistic analysis
- Related work and closing remarks
Evaluating holistic analysis

- Applied both template-only analysis and holistic analysis by hand to three Web benchmark applications
  - Conservatively applied compiler analyses
    - Well-suited to Java (no pointers)
  1. Statically count # of possibly-dependent query-update template pairs
  2. Measure consistency traffic for a scalable query-caching system
The Ferdinand query cache system

• Each proxy server has both a local cache and part of a distributed cache
Up to 40% static improvement over template-only analysis

![Bar chart showing improvements over different types of analysis.](image-url)
Holistic analysis eliminates up to 15% of consistency messages
Most improvement due to existing-primary-key optimization

![Bar graph showing notifications per update for different applications and optimizations.](image-url)
Scalable consistency matters

![Graph showing throughput (WIPS) for different activities with two categories: Broadcast and Ferdinand. Activities include bookstore browsing mix, bookstore shopping mix, auction, and bulletin board.](image-url)
Talk outline

• The DB cache update problem and holistic application analysis
• Implementing holistic analysis
  – Unused-data optimization
  – Existing-primary-key optimization
• Evaluating holistic analysis
  → Related work and closing remarks
Related work

• Data dependence analysis
  – Object-oriented DB, persistent programming languages [many]
  – Object-relation mapping tools [Hibernate, TopLink]
  – Programmer-specified data mapping [Mehlnik+ SIGMOD’07]

• Holistic analysis for latency reduction [Manjhi ‘08]
Holistic analysis conclusions

• Significant extension over traditional database-only techniques
  – Reduces consistency traffic over those techniques

• Limited implementation and evaluation here
  – Generalizes to other languages

• Generalizes to other problems?