StatusQuo: Making Familiar Abstractions Perform Using Program Analysis

Alvin Cheung
Samuel Madden
Armando Solar-Lezama
MIT

Owen Arden
Andrew C. Myers
Cornell
Developing Database Applications

Application Server

Java

SQL Database

Query
Developing Database Applications

Language Choice for Application Logic

Program analysis to the rescue!

Application Distribution

Application Server

SQL Database
• Express application logic in ways that programmers are comfortable with

• Job of compiler & runtime to determine the most efficient implementation
Two Key Technologies

• Infer queries from imperative code

• Migrate computation between servers for optimal performance
List getUsersWithRoles () {
    List users = getUsersFromDB();
    List roles = getRolesFromDB();
    List results = new ArrayList();
    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id) {
                results.add(u);
            }
        }
    }
    return results;
}
Relational Operations in Imperative Code

List getUsersWithRoles () {
    List users = getUsersFromDB();
    List roles = getRolesFromDB();
    List results = new ArrayList();
    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id)
                results.add(u); }
    }
    return results; }

List getUsersWithRoles () {
    return executeQuery(  
        "SELECT u FROM users u, roles r
         WHERE u.roleId == r.id  
          ORDER BY u.roleId, r.id"; }

Goal
Find a variable that we can rewrite into a SQL expression

post-condition variable
Query By Synthesis (QBS)

• Identify potential code fragments
  – i.e., regions of code that fetches persistent data and return values

• Find SQL expressions for post-condition variables

• Try to prove that those expressions preserve program semantics
  – if so, convert the code!
Initial Code Fragments Identification

• Find program points that retrieve persistent data

• Run an inter-procedural analysis that:
  – determine where persistent data are used
  – delimit code fragment to analyze
Search for Post-Condition Expressions

List getUsersWithRoles () {
    List users = query(select * from users);
    List roles = query(select * from roles);
    List results = [];
    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id)
                results = results + []
        }
    }
    return results;
}

Relations involved:
users, roles

Possible expressions to consider for results:
\[\sigma_f(users)\quad top_f(users)\quad \pi_f(users \bowtie_g roles)\]
\[\pi_f(\sigma_g(users) \bowtie_h roles)\quad \text{other expressions involving users, roles}\]

Infinite search space size!
List getUsersWithRoles () {
    List users = query(select * from users);
    List roles = query(select * from roles);
    List results = [];
    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id)
                results = results : []
        }
    }
    return results; }

If outer loop invariant is true and outer loop terminates then post-condition expression is true

Still need a smarter way to search
Search for Post-Condition Expressions and Invariants

- Use **program synthesis** as search engine

Symbolic desc. of search space

Solution constraints

**Program synthesizer**

Expression that satisfies all the constraints

Symbolic manipulation

Counter-example driven search
Experiments
## Real-world Evaluation

**Wilos (project management application) – 62k LOC**

<table>
<thead>
<tr>
<th>Operation type</th>
<th># Fragments found</th>
<th># Fragments converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projection</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Selection</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>Join</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Aggregation</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>33</strong></td>
<td><strong>28</strong></td>
</tr>
</tbody>
</table>
Performance Evaluation: Join Query

Nested-loop join \( \rightarrow \) Hash join!

- Original (lazy): \( O(n^2) \)
- Inferred (lazy): \( O(n) \)

Execution time (ms)

Number of roles / users in DB
Developing Database Applications

Application Logic
- Java

Query
- SQL

Stored Procedures
- PL/SQL

Application Distribution
- Application Server
- SQL Database
discount = executeQuery("select discount from customers
    where id = " + cid);

totalAmount = orderTotal * (1 - discount);

credit = executeQuery("select credit from customers
    where id = " + cid);

if (credit < totalAmount)
    printToConsole("Only " + credit + " in account!\n");
else
    executeUpdate("update customer set credit = " +
        (credit - totalAmount) + " where id = " + cid);
Actual Execution

discount = executeQuery("select discount from customers
    where id = " + cid);

totalAmount = orderTotal * (1 - discount);

credit = executeQuery("select credit from customers
    where id = " + cid);

if (credit < totalAmount)
    printToConsole("Only " + credit + " in account!");
else
    executeUpdate("update customer set credit = " +
        (credit - totalAmount) + " where id = " + cid);
Actual Execution

discount = executeQuery("select discount from customers where id = "+ cid);

totalAmount = orderTotal * (1 - discount);

credit = executeQuery("select credit from customers where id = "+ cid);

if (credit < totalAmount)
    printToConsole("Only "+ credit + " in account!");
else
    executeUpdate("update customer set credit = "+(credit - totalAmount)+" where id = "+ cid);
discount = executeQuery("select discount from customers
where id = " + cid);

totalAmount = orderTotal * (1 - discount);

credit = executeQuery("select credit from customers
where id = " + cid);

if (credit < totalAmount)
    printToConsole("Only " + credit + " in account!");
else
    executeUpdate("update customer set credit = " +
                   (credit - totalAmount) + " where id = " + cid);
discount = executeQuery("select discount from customers where id = " + cid);

totalAmount = orderTotal * (1 - discount);

credit = executeQuery("select credit from customers where id = " + cid);

if (credit < totalAmount)
   printToConsole("Only " + credit + " in account!");
else
   executeUpdate("update customer set credit = " + (credit - totalAmount) + " where id = " + cid);
Speeding up Execution

```java
discount = executeQuery("select discount from customers where id = " + cid);

totalAmount = orderTotal * (1 - discount);

credit = executeQuery("select credit from customers where id = " + cid);

if (credit < totalAmount)
    printToConsole("Only " + credit + " in account!");
else
    executeUpdate("update customer set credit = " + (credit - totalAmount) + " where id = " + cid);
```
Introducing Pyxis

• “Store-procedurizes” DB apps and pushes computation to the DB

• Adaptively controls the amount of computation pushed to DB for optimal performance

• No programmer intervention required
Using Pyxis
How Pyxis Works

Instrument

App Server

Java

SQL

Java

Partition

Deploy

Monitor

DB Server

control transfer

Java

Java

Java

Java

Java

Java

Java

Java

Java

Java
Generating Program Partitions

• Deploy and profile application as-is
• Construct a dependence graph of program statements
  – captures both control and data flow
• Formulate linear program from profile data and dependence graph
  – solution gives a partitioning of the source code
Executing Partitioned Programs

• Pyxis compiler translates partitioned code into standard Java code
• Pyxis runtime executes compiled Java code
  – runtime is just another Java program running on a standard JVM
  – includes monitoring component to determine partition switching
Experiments
Experiment Setup

• TPC-C Java implementation
  – 20 terminals issuing new order transactions
  – DB server has 16 cores total

– Compared against two implementations:
  • JDBC: everything on app server except for JDBC stmts
  • Manual: custom “store procedurized” implementation where everything is on the DB server
All Cores Available

Pyxis generated implementation:
3x latency reduction
1.7x throughput increase
StatusQuo

Ease DB application development

Convert imperative program statements into declarative SQL

Fully automatic code partitioning using application and server characteristics

db.csail.mit.edu/statusquo