DBRIDGE: A program rewrite tool for set-oriented query execution

Mahendra Chavan*, Ravindra Guravannavar, Prabhas Kumar Samanta, Karthik Ramachandra, S Sudarshan

Indian Institute of Technology Bombay,
Indian Institute of Technology Hyderabad

*Current Affiliation: Sybase Inc.
**The Problem**

- Applications often invoke
- Database queries/Web Service requests
  - repeatedly (with different parameters)
  - synchronously (blocking on every request)
- Naive iterative execution of such queries is **inefficient**
  - No sharing of work (eg. Disk IO)
  - Network round-trip delays

The problem is **not** within the database engine!
The problem is **the way queries are invoked** from the application!!

Query optimization:
time to think out of the box
Our Work 1: Batching

Rewriting Procedures for Batched Bindings
Guravannavar et. al. VLDB 2008

- Repeated invocation of a query automatically replaced by a single invocation of its batched form.
- Enables use of efficient set-oriented query execution plans
- Sharing of work (eg. Disk IO) etc.
- Avoids network round-trip delays

Approach

- Transform imperative programs using equivalence rules
- Rewrite queries using decorrelation, APPLY operator etc.
Revised synchronous invocation of queries automatically replaced by asynchronous submission.

Application can perform other work while query executes.

Sharing of work (e.g., Disk IO) on the database engine.

Reduces impact of network round-trip delays.

Extends and generalizes equivalence rules from our VLDB 2008 paper on batching.
**DBridge: Bridging the Divide**

- A tool that implements these ideas on Java programs that use JDBC
  - Set-oriented query execution
  - Asynchronous Query submission
- Two components:
  - **The DBridge API**
    - Handles query rewriting and plumbing
  - **The DBridge Transformer**
    - Rewrites programs to optimize database access
- Significant performance gains on real world applications
The DBridge API

- Java API which extends the JDBC interface, and can wrap any JDBC driver
- Can be used with:
  - Manual writing/rewriting
  - Automatic rewriting (by DBridge transformer)
- Same API for both batching and asynchronous submission
- Abstracts the details of
  - Parameter batching and query rewrite
  - Thread scheduling and management
The DBridge API

```java
stmt = con.prepareStatement("SELECT count(partkey) " +
"FROM part " +
"WHERE p_category=?");

while(!categoryList.isEmpty()) {
    category = categoryList.next();
    stmt.setInt(1, category);
    ResultSet rs = stmt.executeQuery();
    rs.next();
    int count = rs.getInt("count");
    sum += count;
    print(category + ": " + count);
}
```

```java
stmt = con.dbridgePrepareStatement("SELECT count(partkey) " +
"FROM part " +
"WHERE p_category=?");
LoopContextTable lct = new LCT();
while(!categoryList.isEmpty()) {
    LoopContext ctx = lct.createContext();
    category = categoryList.next();
    stmt.setInt(1, category);
    ctx.setInt("category", category);
    stmt.addBatch(ctx);
}
stmt.executeBatch();

for (LoopContext ctx : lct) {
    category = ctx.getInt("category");
    ResultSet rs = stmt.getResultSet(ctx);
    rs.next();
    int count = rs.getInt("count");
    sum += count;
    print(category + ": " + count);
}
```

Before

After
DBRIDGE API – Set oriented execution

```java
LoopContextTable lct = new LoopContextTable();
while(!categoryList.isEmpty()){
    LoopContext ctx = lct createContext();
    category = categoryList.next();
    stmt.setInt(1, category);
    ctx.setInt("category", category);
    stmt.addBatch(ctx);
}
stmt.executeBatch();
for (LoopContext ctx : lct) {
    category = ctx.getInt("category");
    ResultSet rs = stmt.getResultSet(ctx);
    rs.next();
    int count = rs.getInt("count");
    sum += count;
    print(category + "": " + count);
}
```

- `addBatch(ctx)` – insert tuple to parameter batch
- `executeBatch()` – execute set-oriented form of query
- `getResultSet(ctx)` – retrieve results corresponding to the context
LoopContextTable lct = new LoopContextTable();

while(!categoryList.isEmpty()){
    LoopContext ctx = lct createContext();
    category = categoryList.next();
    stmt.setInt(1, category);
    ctx.setInt("category", category);
    stmt.addBatch(ctx);
}

stmt.executeBatch();

for (LoopContext ctx : lct) {
    category = ctx.getInt("category");
    ResultSet rs = stmt.getResultSet(ctx);
    rs.next();
    int count = rs.getInt("count");
    sum += count;
    print(category + " : " + count);
}

- addBatch(ctx) – submits query and returns immediately
- getResultSet(ctx) – blocking wait
DBridge - Transformer

- Java source-to-source transformation tool
- Rewrites programs to use the DBridge API
- Handles complex programs with:
  - Conditional branching (if-then-else) structures
  - Nested loops
- Performs statement reordering while preserving program equivalence
- Uses SOOT framework for static analysis and transformation (http://www.sable.mcgill.ca/soot/)
DBRIDGE - TRANSFORMER

Source Java File → Parsing and Conversion to Interm Rep → Intermediate Code (Jimple) → Dataflow Analysis → Def–Use Information → DDG Construction

Target Java File → Decompile → Modified Jimple Code → Apply Trans Rules → Dependence Graph
Batching: Performance Impact

- Category hierarchy traversal (real world example)
- For small no. of iterations, no change observed
- At large no. of iterations, factor of 8 improvement
Auction system benchmark application
For small no. (4-40) iterations, transformed program slower
At 400-40000 iterations, factor of 4-8 improvement
Similar for warm and cold cache
COMPARISON: BATCHING VS. ASYNCHRONOUS SUBMISSION

- Auction system benchmark application
- Asynchronous execution with 10 threads
CONCLUSIONS AND ONGOING WORK

- Significant performance benefits possible by using batching and/or asynchronous execution for
  - Repeated database access from applications
  - Repeated access to Web services
- DBridge: batching and asynchronous execution made easy
  - API + automated Java program transformation
- Questions? Contact us at
  - http://www.cse.iitb.ac.in/infolab/dbridge
  - Email: karthikksr@cse.iitb.ac.in
Input: A Java Program which uses JDBC

```java
PreparedStatement stmt = con.prepareStatement(
    "SELECT COUNT(p_partkey) AS itemCount
    FROM newpart
    WHERE p_category = ?");

while(category != 0){
    stmt.setInt(1, category);
    ResultSet rs = stmt.executeQuery();
    rs.next();
    int itemCount = rs.getInt("itemCount");
    sum = sum + itemCount;
    category = getParent(category);
}
```
TRANSFORMATION WALK-THROUGH

Step 1 of 5: Identify candidates for set-oriented query execution:

```java
PreparedStatement stmt = con.prepareStatement(
    "SELECT COUNT(p_partkey) AS itemCount
    FROM part
    WHERE p_category = ?");

while(category != 0){
    stmt.setInt(1, category);
    ResultSet rs = stmt.executeQuery();
    rs.next();
    int itemCount = rs.getInt("itemCount");
    sum = sum + itemCount;
    category = getParent(category);
}
```

Iterative execution of a parameterized query

Intention: Split loop at this point
TRANSFORMATION WALK-THROUGH

Step 2 of 5: Identify dependencies that prevent loop splitting:

```java
PreparedStatement stmt = con.prepareStatement(
   "SELECT COUNT(p_partkey) AS itemCount
   FROM part
   WHERE p_category = ?");

while(category != null) {
   stmt.setInt(1, category);
   ResultSet rs = stmt.executeQuery();
   rs.next();
   int itemCount = rs.getInt("itemCount");
   sum = sum + itemCount;
   category = getParent(category);
}
```

Iterative execution of a parameterized query

A Loop Carried Flow Dependency edge crosses the query execution statement
TRANSFORMATION WALK-THROUGH

Step 3 of 5: Reorder statements to enable loop splitting

```java
PreparedStatement stmt = con.prepareStatement("SELECT COUNT(p_partkey) AS itemCount
FROM part
WHERE p_category = ?");

while(category != null){
    int temp = category;
    category = getParent(category);
    stmt.setInt(1, temp);
    ResultSet rs = stmt.executeQuery();
    rs.next();
    int itemCount = rs.getInt("itemCount");
    sum = sum + itemCount;
}
```

Move statement above the Query invocation

Loop can be safely split now
Step 4 of 5: Split the loop (Rule 2)

```
LoopContextTable lct = new LoopContextTable();
while(category != null){
    LoopContext ctx = lct createContext();
    int temp = category;
    category = getParent(category);
    stmt.setInt(1, temp);
    stmt.addBatch(ctx);
}
stmt.executeBatch();

for (LoopContext ctx : lct) {
    ResultSet rs = stmt.getResultSet(ctx);
    rs.next();
    int itemCount = rs.getInt("itemCount");
    sum = sum + itemCount;
}
```
Step 5 of 5: Query Rewrite

```
SELECT COUNT(p_partkey) AS itemCount
FROM part
WHERE p_category = ?
```

```
CREATE TABLE BATCHTABLE1(
  paramcolumn1 INTEGER, loopKey1 INTEGER)

INSERT INTO BATCHTABLE1 VALUES(..., ...)

SELECT BATCHTABLE1.*, qry.*
FROM BATCHTABLE1 OUTER APPLY (
  SELECT COUNT(p_partkey) AS itemCount
  FROM part
  WHERE p_category = paramcolumn1) qry
ORDER BY loopkey1
```

Original Query

Temp table to store Parameter batch

Batch Inserts into Temp table

Set-oriented Query