Automatic Partitioning of Database Applications

Alvin Cheung
Samuel Madden
MIT

Owen Arden
Andrew C. Myers
Cornell
• Database applications are everywhere
  – Basically all web apps

• Why is writing efficient database applications so difficult?
  – Let’s first review a typical architecture
Architecture of DB Applications

App

App Server

Database Queries

DB Server

Java
SQL
Java
SQL
Java
SQL
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);
Speeding up DB Apps: Take 1

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);

8/29/2012
speeding up DB Apps: Take 1

```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);
```
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);
```
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

Deploy

Monitor

Partition

control transfer

Java

Java

Java

Java

Java

Java

Java

Java

Java

Java

Java

SQL

SQL

SQL

SQL

SQL

SQL

SQL

SQL
How Pyxis Works

Instrument → App Server → DB Server

Java

Partition

control transfer

Deploy

Monitor

Java

SQL

Java

SQL

Java

SQL

Java

SQL

Java

SQL

Java
Source Code Partitioning
Application Profiling

- Automatically instrument source code to count the number of times each statement was executed for a short period of time
- Measure capabilities of the application and DB servers

```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);
```
Control Dependencies

- Create program dependence graph that describes control flow and links up variable definitions and uses

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

totalAmount = orderTotal * (1 - discount);

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

credit < totalAmount

printToConsole("Only " + credit + " in account!");

executeUpdate("update customer set credit = " +
    (credit - totalAmount) + " where id = " + cid);
```
Node Weights

- Create program dependence graph that describes control flow and links up variable definitions and uses

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

totalAmount = orderTotal * (1 - discount);

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

credit < totalAmount

printToConsole("Only " + credit + " in account!");

executeUpdate("update customer set credit = " + (credit - totalAmount) + " where id = " + cid);
```
Data Dependencies

- Create program dependence graph that describes control flow and links up variable definitions and uses

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

totalAmount = orderTotal * (1 - discount);

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

credit < totalAmount

printToConsole("Only " + credit + " in account!");

executeUpdate("update customer set credit = " +
(credit - totalAmount) + " where id = " + cid);
```
Data Dependencies

- Create program dependence graph that describes control flow and links up variable definitions and uses

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

totalAmount = orderTotal * (1 - discount);

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

credit < totalAmount

executeUpdate("update customer set credit = " + (credit - totalAmount) + " where id = " + cid);

printToConsole("Only " + credit + " in account!");
```
Minimize: \[ \sum_{e_i \in \text{edges}} e_i \cdot w_{e_i} \]

Subject to: \[ e_i = 1 \; \text{iff} \; n_i \neq n_j \] where \( e_i \) connects \( n_i \) and \( n_j \), and 0 otherwise.

\[ \sum_{n_i \in \text{nodes}} n_i \cdot w_{n_i} \leq \text{budget} \]

Solution
\[ n_i = 0 \rightarrow \text{APP} \]
\[ n_i = 1 \rightarrow \text{DB} \]

Represented using Pyxil (PYXis Intermediate Language)
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);
Pyxil Compilation & Runtime
Pyxil Compiler and Runtime

• Compiles pyxil program into two Java programs
  – To be executed by the Pyxis runtimes on the app and DB server

• Pyxis runtime is simply a Java program running on a standard JVM on the two servers
Pyxil Compilation

• Two issues:
  – Control transfer implementation
  – Heap Synchronization
Control Transfer and Heap Sync

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

:D: totalAmount = orderTotal * (1 - discount);

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

if (:D: credit < totalAmount)
  :A: printToConsole("Only " + credit + " in account!");
else
  :D: executeUpdate("update customer set credit = "+
    (credit - totalAmount) +
    " where id = " + cid);
```
Control Transfer and Heap Sync

```
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!");
executeUpdate("update customer set credit = " + (credit - totalAmount) + " where id = " + cid);
```

Diagram:
- App Server
- DB Server
- Monitor
- Control Flow Diagram

Entry: cid, orderTotal

Credit:
- Credit calculation
- Comparison with total amount
- Console print
- Database update

Entry: cid, orderTotal
Experiments
Experiment Setup

• TPC-C Java implementation
  – 20 terminals issuing new order transactions
  – 0.2ms RTT between app and DB servers
  – 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
Limited Cores Available

Pyxis automatically reduces to JDBC when resources are limited.
Dynamic Switching

Pyxis automatically switches to the most efficient implementation based on current server load.
Pyxis

Ease DB application development

Fully automatic code partitioning using application and server characteristics

Adaptive optimization based on server load

db.csail.mit.edu/pyxis