Rethinking the Application-Database Interface

Alvin Cheung
MIT CSAIL || Univ of Washington
Akamai Study on Travel Site Performance

Loading time – 57% of online shoppers will wait 3 seconds or less before abandoning the site.

MARISSA MAYER @ WEB 2.0

Half a second delay caused a 20% drop in traffic.
Half a second delay killed user satisfaction.

IT now 10 percent of world's electricity consumption
New analysis finds IT power suck has eclipsed aviation

The Register, August 2013
Program logic

Framework

Runtime

OS & Networking

Persistent storage
Program logic

Framework

Runtime

OS & Networking

Persistent storage

StatusQuo: Making Familiar Abstractions Perform Using Program Analysis
CIDR 13 [Best Paper Award]
Seamless transformation of code representation

How

WHERE

Automatic computation migration for best performance
How

WHERE
List getUsersWithRoles () {
    List users = this.userDao.getUsers();
    List roles = this.roleDao.getRoles();
    List results = new ArrayList();
    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id)
                results.add(u);
        }
    }
    return results;
}
List users = this userDao.getUsers();

List getUsers () {
    List users = this.getHibernateTemplate().loadAll(User.class);
    return users;
}

public List org.hibernate.impl.CriteriaImpl.list() {
    return session.list( this );
}

public List org.hibernate.impl.SessionImpl.list (Criteria criteria) {
    checkTransactionSynchStatus();
    String[] implementors = factory.getImplementors(...);
    int size = implementors.length;
    CriteriaLoader[] loaders = new CriteriaLoader[size];
    Set spaces = new HashSet();
    for (int i=0; i<size; i++) {
        loaders[i] = new CriteriaLoader(
            getOuterJoinLoadable( implementors[i], ... );
        spaces.addAll( loaders[i].getQuerySpaces() );
    }
    autoFlushIfRequired(spaces);
    List results = Collections.EMPTY_LIST;
    boolean success = false;
    try {
        for (int i=0; i<size; i++) {
            final List currentResults = loaders[i].list(this);
            currentResults.addAll(results);
            results = currentResults;
        }
        success = true;
    }
    finally {
        dontFlushFromFind--;
        afterOperation(success);
    }
    return results;
}

users = executeQuery("SELECT * FROM users");
List getUsersWithRoles () {
    List users = this.userDao.getUsers();
    List roles = this.roleDao.getRoles();
    List results = new ArrayList();
    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id)
                results.add(u);
        }
    }
    return results;
}

executeQuery("SELECT * FROM Users");
executeQuery("SELECT * FROM Roles");

List getUsersWithRoles () {
    return executeQuery("SELECT u FROM users u, roles r WHERE u.roleId == r.id ORDER BY u.roleId, r.id");
}
List getUsersWithRoles () {
    List users = this.userDao.getUsers();
    List roles = this.roleDao.getRoles();
    List results = new ArrayList();
    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id)
                results.add(u);
        }
    }
    return results;
}

executeQuery("SELECT * FROM Users");
executeQuery("SELECT * FROM Roles");

executeQuery("SELECT u FROM users u, roles r
              WHERE u.roleId == r.id
              ORDER BY u.roleId, r.id");
QUERY By SYNTHESIS

Seamless transformation of code representation

Optimizing Database-Backed Applications with Query Synthesis
PLDI 13
### Wilos (project management application) – 62k LOC

<table>
<thead>
<tr>
<th># fragments found</th>
<th># fragments converted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>33</strong></td>
<td><strong>29</strong></td>
</tr>
</tbody>
</table>

### iTacker (bug tracking system) – 61k LOC

<table>
<thead>
<tr>
<th># fragments found</th>
<th># fragments converted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>16</strong></td>
<td><strong>12</strong></td>
</tr>
</tbody>
</table>

---

**Graph:**

- **X-axis:** Number of roles/users in DB (0 to 100K)
- **Y-axis:** Page load time (ms) (0 to 1000K)
- **Legend:**
  - *original* (blue line)
  - *QBS converted* (red line)
List getUsersWithRoles () {
    List users = this.userDao.getUsers();
    List roles = this.roleDao.getRoles();
    List results = new ArrayList();
    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id)
                results.add(u);
        }
    }
    return results; }  // output variable

List getUsersWithRoles () {
    return executeQuery("SELECT u FROM users u, roles r WHERE u.roleId == r.id ORDER BY u.roleId, r.id"); }  // output variable
List getUsersWithRoles () {
    List users = this.userDao.getUsers();
    List roles = this.roleDao.getRoles();
    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"); }

output variable
List getUsersWithRoles () {
    List users = this.userDao.getUsers();
    List roles = this.roleDao.getRoles();
    List results = new ArrayList();

    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id) {
                results.add(u);
            }
        }
    }

    results = \pi_{user}( users \bowtie_{roleId=id} roles )  
    return results;
}

Hoare-style program verification
for (int i = 0; i < users.size(); ++i) {
    for (int j = 0; j < roles.size(); ++j) {
        if (users[i].roleId == roles[j].id)
            results.add(users[i]);
    }
    results[i+1] = results[i] \land\!
    users[i] \Join_{roleId=id} roles \rightarrow \text{post-condition (program state}_\text{final})
}

\Join: list / element join
\pi: projection of fields

Verification Conditions

Invariant is true on loop entry:
first enter loop \rightarrow \text{invariant (program state}_\text{initial})

Invariant is preserved:
loop continues \land \text{invariant (program state}_i \rightarrow \text{invariant (program state}_i+1)

Invariant implies post-condition on loop exit:
loop terminates \land \text{invariant (program state}_i \rightarrow \text{post-condition (program state}_\text{final})
for (int i = 0; i < users.size(); ++i) {
    for (int j = 0; j < roles.size(); ++j) {
        if (users[i].roleId == roles[j].id)
            results.add(users[i]);
    }
}

result_{i+1} = result_i : users[i] \Join_{roleld = id} roles → post-condition (program state final)

Invariant is preserved:
loop continues ∧ invariant (program state \_i) → invariant (program state \_i+1)

\[
\begin{align*}
    j < \text{size(roles)} & \land \text{ result}_{i+1} = \text{result}_i : \text{users}[i] \Join_{roleld = id} \text{roles} [0..j] \\
    \[ \pi_{id}(\text{roles}[j]) = \pi_{roleld}(\text{users}[i]) \land \text{invariant (program state } i+1) \] \\
    \lor \\
    \[ \pi_{id}(\text{roles}[j]) \neq \pi_{roleld}(\text{users}[i]) \land \text{invariant (program state } i+1) \]
\end{align*}
\]
for (int i = 0; i < users.size(); ++i) {
    for (int j = 0; j < roles.size(); ++j) {
        if (users[i].roleId == roles[j].id)
            results.add(users[i]);
    }

    results_{i+1} = results_{i} : users[i] \Join_{\text{roleId = id}} roles \rightarrow \text{post-condition (program state }_{\text{final}}) 

Invariant is preserved:

\text{loop continues } \land \text{ invariant (program state }_{i}) \Rightarrow \text{ invariant (program state }_{i+1})

j < \text{size( roles ) } \land results_{j+1} = results_{j} : users[i] \Join_{\text{roleId = id}} roles [0..j] \Rightarrow

[ \pi_{\text{id}}(\text{roles}[j]) = \pi_{\text{roleId}}(users[i]) \land results : users[i] = ... ]
\lor

[ \pi_{\text{id}}(\text{roles}[j]) \neq \pi_{\text{roleId}}(users[i]) \land \text{ invariant (program state }_{i+1}) ]
Invariant is preserved:

loop continues $\land$ invariant (program state $i$) $\Rightarrow$ invariant (program state $i+1$)

\[
\begin{align*}
    j < \text{size}(\text{roles}) & \land \quad \text{results}_{j+1} = \text{results}_j : \text{users}[i] \Join_{\text{roleId} = \text{id}} \text{roles}[0..j] \\
    [\pi_{\text{id}}(\text{roles}[j]) = \pi_{\text{roleId}}(\text{users}[i]) & \land \quad \text{results} : \text{users}[i] = \\
    \land \quad \text{results} : \text{users}[i] \Join_{\text{roleId} = \text{id}} \text{roles}[0..j+1] ] \\
    \lor \\
    [\pi_{\text{id}}(\text{roles}[j]) \neq \pi_{\text{roleId}}(\text{users}[i]) & \land \quad \text{invariant (program state } i+1) ]
\end{align*}
\]
for (int i = 0; i < users.size(); ++i) {
    for (int j = 0; j < roles.size(); ++j) {
        if (users[i].roleId == roles[j].id)
            results.add(users[i]);
    }

    invariant (program state \(i\))

    results_{j+1} = results_j : users[i] \Join_{roleId = id} roles \[0..j\] →
    post-condition (program state \(final\))
}

Invariant is preserved:

loop continues \(\land\) invariant (program state \(i\)) \(\rightarrow\) invariant (program state \(i+1\))

\(j < \text{size(roles)} \land\) results_{j+1} = results_j : users[i] \Join_{roleId = id} roles \[0..j\] \(\rightarrow\)

\([\pi_{id}(\text{roles}[j]) = \pi_{roleId}(\text{users}[i]) \land\) results : users[i] =

\([\pi_{id}(\text{roles}[j]) \neq \pi_{roleId}(\text{users}[i]) \land\) results = ...
}
for (int i = 0; i < users.size(); ++i) {
    for (int j = 0; j < roles.size(); ++j) {
        if (users[i].roleId == roles[j].id)
            results.add(users[i]);
    }
    results_{i+1} = results_{i} : users[i] \times_{roleId = id} roles_{[0..j]} → post-condition (program state final)
}

Invariant is preserved: ✓

loop continues \land invariant (program state _i_) \rightarrow invariant (program state _i+1_)

\[ j < size(\ roles ) \land results_{j+1} = results_{j} : users[i] \times_{roleId = id} roles_{[0..j]} \rightarrow \]

\[ \left[ \pi_{id}(roles[j]) = \pi_{roleId}(users[i]) \land results : users[i] = results : users[i] \times_{roleId = id} roles_{[0..j+1]} \right] \lor \]

\[ \left[ \pi_{id}(roles[j]) \neq \pi_{roleId}(users[i]) \land results = results : users[i] \times_{roleId = id} roles_{[0..j+1]} \right] \]
for (int i = 0; i < users.size(); ++i) {
    for (int j = 0; j < roles.size(); ++j) {
        if (users[i].roleId == roles[j].id)
            results.add(users[i]);
    }
}

results_{i+1} = results_i \uplus roles \rightarrow \text{post-condition (program state } \text{final})
Hoare style reasoning

loop continues ∧ invariant (program state \(i\)) → invariant (program state \(i+1\))

…

+ Relational algebra

\(\sigma, \bowtie, \pi, \ldots\)

‖

Theory of ordered relations
Theory of ordered relations

Similar to relational algebra
Model relations as ordered lists

\[
L := \text{program var} \quad e := L[i]
\]

\[
\begin{align*}
&| [] & | e \ op \ e \\
&| L : L & | \max(L) \ | \min(L) \\
&| L[0..e] & | \sum(L) \ | \avg(L) \\
&| L \times_f L & | \size(L) \\
&| \pi_f(L) & | \order_e(L)
\end{align*}
\]
Theory of ordered relations

Similar to relational algebra
Model relations as ordered lists

\[
L[0..i] \begin{cases} 
  [][0..i] &= [] \\
  L[0..i] &= [] & \text{if } i = 0 \\
  ([e]:L)[0..i] &= [e]:L[0..i-1] & \text{if } i > 0
\end{cases}
\]
List getUsersWithRoles ()
{
    List users = this.userDao.getUsers();
    List roles = this.roleDao.getRoles();
    List results = new ArrayList();

    for (User u : users)
    {
        for (Role r : roles)
        {
            if (u.roleId == r.id)
            {
                results.add(u);
            }
        }
    }
    return results;
}
List getUsersWithRoles ()
{
    List users = this.userDao.getUsers();
    List roles = this.roleDao.getRoles();
    List results = new ArrayList();

    for (User u : users)
    {
        for (Role r : roles)
        {
            if (u.roleId == r.id)
            {
                results.add(u);
            }
        }
    }

    return results;
}
List getUsersWithRoles ()
{
    List users = this.userDao.getUsers();
    List roles = this.roleDao.getRoles();
    List results = new ArrayList();

    for (User u : users)
    {
        for (Role r : roles)
        {
            if (u.roleId == r.id)
            {
                results.add(u);
            }
        }
    }

    return results;
}
Program synthesizer

Search space description

Solution constraints

Expression that satisfies constraints

Scan
Find
Verify
Program synthesizer

Search space description

Verification conditions:
- Invariant is true initially
- Invariant is preserved
- Invariant implies post condition

Expression that satisfies constraints
List getUsersWithRoles() {
    List users = this.userDao.getUsers();
    List roles = this.roleDao.getRoles();
    List results = new ArrayList();

    for (User u : users) {
        for (Role r : roles) {
            if (u.roleId == r.id) {
                results.add(u);
            }
        }
    }

    return results;
}
Program synthesizer

Expression grammar

\[ \sigma_f(\text{users}) \quad \text{users}[0..e] \]
\[ \pi_f(\text{users} \land g \text{ roles}) \]
\[ \pi_f(\sigma_g(\text{users} \land h \text{ roles}) \ldots) \]

Verification conditions

- Invariant is true initially
- Invariant is preserved
- Invariant implies post condition

Expression that satisfies constraints
Expression grammar

- $\sigma_f(\text{users}) \leq \text{users}[0..e]$
- $\pi_f(\text{users} \times_g \text{roles})$
- $\pi_f(\sigma_g(\text{users} \times_h \text{roles}))$

Verification conditions

- Invariant is true initially
- Invariant is preserved
- Invariant implies post condition

Use the Sketch synthesizer
symbolic manipulation of search space

Scan
Find
Verify

Expressions for invariants and post conditions

- $\text{invariant}(\ldots) = \ldots$
- $\text{post-condition}(\ldots) = \ldots$
Program synthesizer

Expression grammar

\[ \sigma_f(\text{users}) \quad \text{users}[0..e] \]
\[ \pi_f(\text{users} \nabla g \text{ roles}) \]
\[ \pi_f(\sigma_g(\text{users} \nabla h \text{ roles})) \]

Solve incrementally
Increase complexity of expression grammar iteratively

Break symmetries
Use relational equivalences
\[ \sigma_f(\sigma_g(L)) = \sigma_g(\sigma_f(L)) \]

Expression for invariants and post conditions
\[ \text{invariant}(\ldots) = \ldots \]
\[ \text{post-condition}(\ldots) = \ldots \]

Grammar only include one of the expressions

Scan
Find
Verify
Query By Synthesis toolchain

Application source code

Code fragment identifier

Verification conditions computation

Invariant + post-condition synthesizer

Code fragment to be analyzed

Inferred SQL

Transformed method body
Experiments

Two large-scale open source web apps
   Developed by teams of programmers
   Use variety of idioms to access persistent data

Generate benchmarks from code

Goals
   Number of cases successfully converted
   Effect of conversions
### 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>2</td>
<td>2</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>29</strong></td>
</tr>
</tbody>
</table>

### iTracker (bug tracking system) – 61k LOC

<table>
<thead>
<tr>
<th>Operation type</th>
<th># fragments found</th>
<th># fragments converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projection</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Selection</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Join</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Aggregation</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>16</strong></td>
<td><strong>12</strong></td>
</tr>
</tbody>
</table>

Max code fragment size: ~200 lines of code
Max compilation time: 5 minutes
QBS code runs 10x faster for selections

50% selectivity

10% selectivity

Retrieve only necessary data
Push computation to DB

Page load time (ms)

Total number of users in DB
QBS code runs asymptotically faster for joins

Page load time (ms)

Number of roles / users in DB

Nested-loop join $\rightarrow$ Hash join!

Original $O(n^2)$ $\rightarrow$ QBS converted $O(n)$
QBS code runs 100x faster for aggregates

Use database table statistics
Achievement unlocked
Can translate code representations

Achievement unlocked
Can build new classes of compilers
StatusQuo: Making Familiar Abstractions Perform Using Program Analysis
CIDR 13 [Best Paper Award]
How WHERE
discount = query("SELECT... FROM customer");

total = orderTotal * (1 - discount);

credit = query("SELECT... FROM customer");

if (credit < total)
    print("Only: ") + credit + " in acct");
else
    update("UPDATE... SET credit = " + (credit - total));
discount = query("SELECT... FROM customer");

total = orderTotal * (1 - discount);

credit = query("SELECT... FROM customer");

if (credit < total)
    print("Only: " + credit + " in acct");
else
    update("UPDATE... SET credit = " + (credit - total));
discount = query("SELECT... FROM customer");

total = orderTotal * (1 - discount);

credit = query("SELECT... FROM customer");

if (credit < total)
    print("Only: " + credit + " in acct");
else
    update("UPDATE... SET credit = " + (credit - total));

Move computation to the data
discount = query("SELECT... FROM customer");

total = orderTotal * (1 - discount);

credit = query("SELECT... FROM customer");

if (credit < total)
    print("Only: " + credit + " in acct");
else
    update("UPDATE... SET credit = " + (credit - total));
discount = query("SELECT... FROM customer");

total = orderTotal * (1 - discount);

credit = query("SELECT... FROM customer");

if (credit < total)
    print("Only: " + credit + "in acct");
else
    update("UPDATE... SET credit = " + (credit - total));
```java
discount = query("SELECT... FROM customer");

total = orderTotal * (1 - discount);

credit = query("SELECT... FROM customer");

if (credit < total)
    print("Only: " + credit + " in acct");
else
    update("UPDATE... SET credit = " + (credit - total));
```
Pyxis

Automatically pushes code to the database
Adaptively controls the amount of code pushed for optimal performance

Automatic Partitioning of Database Applications
VLDB 12
Speeding up Database Applications with Pyxis
SIGMOD 13
Execution counts

100  discount = query("SELECT... FROM customer");
100  total = orderTotal * (1 - discount);
100  credit = query("SELECT... FROM customer");
100  if (credit < total)
25      print("Only: " + credit + " in acct");
else
75      update("UPDATE... SET credit = " + (credit - total));
Program dependence graph

```java
discount = query("SELECT... FROM customer");

total = orderTotal * (1 - discount);

credit = query("SELECT... FROM customer");

credit < total

print("Only: " + credit + " in acct");

update("UPDATE... SET credit = " + (credit - total));
```
discount = query("SELECT... FROM customer");

total = orderTotal \times (1 - \text{discount});

credit = query("SELECT... FROM customer");

\text{credit < total}

print("Only: " + credit + " in acct");

update("UPDATE... SET credit = " + (credit - total));
update("UPDATE... SET credit = " + (credit - total));

discount = query("SELECT... FROM customer");

total = orderTotal * (1 - discount);

credit = query("SELECT... FROM customer");

credit < total

print("Only: " + credit + " in acct");

updatedb("UPDATE... SET credit = " + (credit - total));
Integer
Linear
Program

Minimize:
cut edge weights

Subject to:
resources on DB
= 100%
Integer Linear Program

Minimize:
cut edge weights

Subject to:
resources on DB
= 100%
Integer Linear Program

Minimize: cut edge weights

Subject to: resources on DB = 25%
Experiments

TPC-C Java implementation
20 concurrent users issuing transactions
16 cores DB, 0.5ms latency between servers

Compared against two implementations

**JDBC**: all code on app server except for JDBC calls

**Manual**: hand-written “stored procedurized” version where most code is on the database server
Pyxis performs like manual when CPUs are available

3x latency reduction
1.7x throughput increase
Pyxis adapts to changing environments

Average Latency (ms)

Time (s)

JDBC
Manual
Pyxis

Limited CPUs available → JDBC

All CPUs available → Manual

Reduced CPUs available

N%: percent of transactions served using JDBC partition
Achievement unlocked
Can move computation for optimal performance
Sloth: Being Lazy is a Virtue (When Issuing Database Queries)

Pyxis

- Achievement unlocked: Can translate code representations
- Achievement unlocked: Can build new classes of compilers
- Achievement unlocked: Can move computation for optimal performance

3x latency reduction
2x throughput increase

Sloth

Achievement unlocked: Can batch queries to reduce roundtrips

PLDI 13
VLDB 12
SIGMOD 13

POPL 14, SIGMOD 14
Related Work

[Grust 09] FERRY: database-supported program execution
[Iu 10] HadoopToSQL: a MapReduce query optimizer
[Meijer 06] LINQ: Reconciling objects, relations and XML in the .NET framework
[Srivastava et al 09] Program verification using templates over predicate abstraction
[Tatlock 10] Bringing extensibility to verified compilers
[Wiedermann 07] Interprocedural query extraction for transparent persistence

[Chong 07] Secure web applications via automatic partitioning
[Chun 11] CloneCloud: elastic execution between mobile device and cloud
[Cuervo 10] Maui: making smartphones last longer with code offload
[Newton 09] Wishbone: Profile-based partitioning for sensornet applications
[Ra 11] Odessa: enabling interactive perception applications on mobile devices
[Yang 07] A unified platform for data driven web applications with automatic client-server partitioning
StatusQuo: Making Familiar Abstractions Perform Using Program Analysis
CIDR 13 [Best Paper Award]
GraphLab
Spark
Oracle
SQL Server
MySQL
Sybase
IBM DB2
PostgresQL
Not Only SQL
VOLTDB
Clustrix
xeround
nuodb
memsql
Intel Quad-Core Processor
Optical FPGA
Query

Parser

Optimizer

Executor

Algebraic rewrites
Access path selection
Statistics based optimizations
Query

- Parser
- Optimizer
- Executor

Algebraic rewrites
Access path selection
Statistics-based optimizations
Application

Query

Application specific storage

High-level storage specification

Systems Synthesis Analysis Languages
Rethinking the Application-Database Interface

Alvin Cheung
akcheung@cs.washington.edu