Unifying Remote Data, Remote Procedures, and Service Clients

William R. Cook
University of Texas at Austin
with
Eli Tilevich, Yang Jiao, Virginia Tech
Ali Ibrahim, Ben Wiedermann, UT Austin
RPC

Ease of Programming

Transactionally Efficient

SQL

Web Services

Cross-platform
Latency tolerant

Ease of Programming
“Whatever the database programming model, it must allow complex, data-intensive operations to be picked out of programs for execution by the storage manager...”

David Maier, DBLP 1987
Language Design: Remote Calls

Starting point

\[
\begin{align*}
\text{print}( \ r \text{.getName}() \ ); \\
\text{print}( \ r \text{.getSize}() \ );
\end{align*}
\]

Goals

Fast: one round trip
Stateless communication
\[ \text{do not require persistent connection} \]
Platform independent
\[ \text{no serialization of complex user-defined objects} \]
Clean programming model

Notes:

\[
\begin{align*}
\text{print is local} \\
r \text{ is remote}
\end{align*}
\]
A Novel Solution: Batches

```java
batch ( Item r : service ) {
    print( r.getName() );
    print( r.getSize() );
}
```

**Execution model: Batch Command Pattern**

1. Client *sends script* to the server
   (Creates Remote Façade on the fly)
2. Server *executes* two calls
3. Server *returns results in bulk* (name, size)
   (Creates Data Transfer Objects on the fly)
4. Client *runs the local code* (print statements)
// create remote script
script = <[
    out_A( *.getName() )
    out_B( *.getSize() )
]>;
// execute on the server
Forest x = service.execute( script );

// Client uses the results
print( x.get("A") );
print( x.getInt("B") );
A larger example

```
int limit = ...;
Service<Mailer> serviceConnection = ...;

batch ( Mailer mail : serviceConnection ) {
    for ( Message msg : mail.Messages )
        if ( msg.Size > limit ) {
            print( msg.Subject & " Deleted" );
            msg.delete();
        }
        else
            print( msg.Subject & ":" & msg.Date );
}
```
Remote part as Batch Script

```plaintext
script = <[
  for ( msg : *.Messages ) {
    out_A( msg.Subject );
    if ( out_B( msg.Size > in_X ) ) {
      msg.delete();
    } else {
      out_C( msg.Date );
    }
  ]>
```
Service<Mailer> serviceConnection = ...;
in = new Forest();
in.put("X", limit);
Forest result =
    serviceConnection.execute(script, in);
for ( r : result.getIteration("msg") )
    if ( r.getBoolean("B") )
        print( r.get("A") & " Deleted" );
    else
        print( r.get("A") & ":" & r.get("C") );
Forest Structure == Control flow

```python
for (x : r.Items) {
    print( x.Name );
    for (y : x.Parts)
        print( y.Name );
}
```

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Parts</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Engine&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Hood&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Wheel&quot;</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Cylinder&quot;</td>
</tr>
<tr>
<td>&quot;Tire&quot;</td>
</tr>
<tr>
<td>&quot;Rim&quot;</td>
</tr>
</tbody>
</table>
```
Batch = One Round Trip

Clean, simple performance model

Some batches would require more round trips

```java
batch (...) {
    if (AskUser("Delete " + msg.Subject + "?"))
        msg.delete();
}
```

Pattern of execution

**OK:** Local → Remote → Local

**Error:** Remote → Local → Remote

Can't just mark everything as a batch!
What about Object Serialization?

Batch only transfers primitive values, not objects
But they work with any object, not just *remotable* ones

Send a local set to the server?

```java
java.util.Set<String> local = ... ;
batch ( mail : server ) {
    service.Set recipients = local; // compiler error
    mail.sendMessage( recipients, subject, body);
}
```
Serialization by Public Interfaces

```
java.util.Set<String> local = ... ;

batch ( mail : server ) {
    service.Set recipients = mail.makeSet();
    for (String addr : local )
        recipients.add( addr );
    mail.sendMessage( recipients, subject, body);
}
```

Sends list of addresses with the batch

Constructors set on server and populates it

**Works between different languages**

Serialization can be encapsulated in a procedure
Batch Summary

Client
Batch statement: compiles to Local/Remote/Local
Works in any language (e.g. Java, Python, JavaScript)
Cross-language and cross-platform

Server
Small engine to execute scripts
Call only public methods/fields (safe as RPC)
Stateless, no remote pointers (aka proxies)

Communication
Forest(s) (trees) of primitive values (no serialization)
Efficient and portable
Batch Script Language

\[ e ::= x \mid c \]
| if e then e else e | variables, constants |
| for x : e do e | conditionals |
| let x = e in e | loops |
| x = e \mid e.x = e | binding |
| e.x | assignment |
| e.m(e, ..., e) | fields |
| e.m(e, ..., e) | method call |
| e \ldots e | primitive operators |
| in_x \mid out_x e | parameters and results |
| fun(x) e | functions |

Agree on script format, not on object representation
SQL

30 years

ODBC
JDO/EJB
LINQ

Database Clients
Call Level Interface (e.g. JDBC)

// create a remote script/query
String q = "select name, size
    from files
    where size > 90";

// execute on server
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(q);

// use the results
while ( rs.next() ) {
    print( rs.getString("name") );
    print( rs.getInteger("size") );
}
Call Level Interface (e.g. JDBC)

```java
// create a remote script/query
String q = "select name, size 
    from files 
    where size > 90";

// execute on server
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(q);

// use the results
while ( rs.next() ) {
    print( rs.getString("name") );
    print( rs.getInteger("size") );
}
```
In-Memory Objects

```java
for ( File f : directory.Files )
    if ( f.Size > 90 ) {
        print( f.Name );
        print( f.Size );
    }
```
**Batches ==> SQL**

```java
batch ( Service<FileSystem> directory : service ) {
    for ( File f : directory.Files )
        if ( f.Size > 90 ) {
            print( f.Name );
            print( f.Size );
        }
}
```

**Batch Script:**

```
< [ for (f : *.Files)
    if (f.Size > 90) { out_A(f.Name); out_B(f.Size) } ] >
```

**SQL:**

```
SELECT f.Name, f.Size
FROM Files
WHERE f.Size > 90
```
Data Schema = Server Interface

```java
public class Northwind {
    Set<Customer> Customers;
    Set<Order> Orders;
    void insertCustomer(Customer c);
    void insertOrder(Order o);
}

@Table(name="Customers")
public class Customer {
    @Id String CustomerID;
    String CompanyName;
    String ContactName;
    String Country;
    @Inverse("Customer")
    Set<Order> Orders;
    delete();
}

@Table(name="Orders")
public abstract class Order {
    @Id public int OrderID;
    public Date OrderDate;
    public Date RequiredDate;
    public Date ShippedDate;
    @Column(name="CustomerID")
    public Customer Customer;
    delete();
}
```
LINQ

// create the remote script/query
var results = from f in files
    where size > 90
    select { f.name, f.size };

// execute and use the results
for (var rs in results) {
    print( rs.name );
    print( rs.size );
}
Dynamic Queries in LINQ

```csharp
var matches = db.Course;
// add a test if the condition is given
if (Test.Length > 0)
    matches = matches.Where(c => c.Title == Test);
// select the desired values
matches = matches.Select(c => c.Title);
// iterate over the result set
for (String title : matches.ToList())
    print(title);
```
Dynamic Queries in Batches

```java
batch (db : Database) {
    for (Ticket t : db.Course)
        if (Test.Length == 0 || c.Title == Test)
            print(c.Title);
}
```

Left side of condition is *client-only*: Pre-evaluated
Modularity

Forested Model (query-based)

- Large Input
- Query
- Subset
- Transform
- Results

Deforested Model (need-based)

- Large Input
- Transform (and select)
- Results
Modularity

Forested Model (query-based)

Large Input → Query2 → Subset2 → Transform2 → Results2

Deforested Model (need-based)

Large Input → Transform2 (and select) → Results
Batches for SQL

Batch compiler creates SQL automatically
  Efficient handling of nested of loops
  Always a constant number of queries for a batch
    No matter how many (nested) loops are used

Supports all aspects of SQL
  Queries, updates, sorting, grouping, aggregations

Summary
  Clean fine-grained object-oriented programming model
  Efficient SQL batch execution
Amazon Web Service

<ItemLookup>
  <AWSAccessKeyId>XYZ</AWSAccessKeyId>
  <Request>
    <ItemIds>
      <ItemId>1</ItemId>
      <ItemId>2</ItemId>
    </ItemIds>
    <IdType>ASIN</IdType>
    <ResponseGroup>SalesRank</ResponseGroup>
    <ResponseGroup>Images</ResponseGroup>
  </Request>
</ItemLookup>
Available Now...

Jaba: Batch Java
  100% compatible with Java 1.5
  Transport: XML, JSON, easy to add more

Batch statement as “for”
  for (RootInterface r : serviceConnection) { ... }

Full SQL generation and ORM
  Select/Insert/Delete/Update, aggregate, group, sorting

Future work
  Security models, JavaScript/Python clients

Edit and debug in Eclipse or other tools
  Available now!
Opportunities

Add batch statement to your favorite language
   Easy with reusable partitioning library
   Scala, C#, Python, JavaScript, COBOL, Ruby, etc...
   Monads?

Optimization by partial evaluation

What about multiple servers in batch?
   Client $\rightarrow$ Server*         Client $\rightarrow$ Server $\rightarrow$ Server
   Client $\leftrightarrow$ Server

Generalize “remoteness”: MPI, GPU, ...

Concurrency, Asynchrony and Streaming
Related work

Microsoft LINQ
Batches are different and more general than LINQ

Mobile code / Remote evaluation
Does not manage returning values to client

Implicit batching
Performance model is not transparent

Asynchronous remote invocations
Asynchrony is orthogonal to batching

Automatic program partitioning
binding time analysis, program slicing

Deforestation
Introduce inverse: *reforestation* for bulk data transfer
Transactionally, Efficient
(POPL 2007, TBD)

RPC

Ease of
Programming
(ECOOP 2009)

batch

SQL

Cross-platform
(ECOWS 2009)

Web Services

Ease of Programming
(ECOOP 2009)
Conclusion

Batch Statement
General mechanism for partitioned computation

Unifies
Distributed objects (RPC)
Relational (SQL) database access
Service invocation (Web services)

Benefits:
Efficient distributed execution
Clean programming model
No explicit queries, stateless, no proxies
Language/transport neutral

Requires adding batch statement to language!