Batches
Unified and Efficient Access to RPC, WS, and SQL Services

William R. Cook
University of Texas at Austin
with
Ben Wiedermann, Harvey Mudd
Eli Tilevich, Virginia Tech
Ali Ibrahim, Google
Advertisement

Enso – interpreted integrated DSLs

Hybrid Partial Evaluation

Orc – structured concurrency

Batches – this talk
Three kinds of “remoteness”
Distributed Objects

CORBA
DCOM
RMI

30 years

proc(data)

RPC
Benefits
Use existing languages
Elegant OO model

Problems
Latency (many round trips)
Stateful communication
Platform-specific

Solutions?
Remote Façade
Data Transfer Object

Solutions are as bad as the problem!
RPC
Ease of Programming

SQL
Transactional
Efficient

Web Services
Cross-platform
Latency tolerant

?
Impedance Mismatch

“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
Start Over: Remote Calls

Starting point

```java
print( r.getName() );
print( r.getSize() );
```

Notes:
- print is local
- r is remote

Goals

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

```java
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

```java
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>“Engine”</td>
<td></td>
</tr>
<tr>
<td>“Hood”</td>
<td></td>
</tr>
<tr>
<td>“Wheel”</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Cylinder”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Tire”</td>
</tr>
<tr>
<td>“Rim”</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 remutable ones

Send a local set to the server?

Set<String> local = ... ;

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

```java
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
Interprocedural Batches

Reusable serialization function

```java
@Batch
service.Set send(Mail server, local.Set<String> local) {
    service.Set remote = server.makeSet();
    for (String addr : local )
        remote.add( addr );
    return remote;
}
```

Main program

```java
batch ( mail : server ) {
    remote.Set recipients = send( localNames );
}
Exceptions

Server Exceptions
   Terminate the batch
   Return exception in forest
   Exception is raised in client at same point as on server

Client Exceptions
   Be careful!
   Batch has already been fully processed on server
   Client may terminate without handling all results locally
Transactions and Partial Failure

Batches are not necessarily transactional
  But they do facilitate transactions
  Server can execute transactionally

Batches reduce the chances for partial failure
  Fewer round trips
  Server operations are sent in groups
Order of Execution Preserved

All local and remote code runs in correct order

```java
batch ( remote : service ) {
    print( remote.updateA( local.getA() ));  // getA, print
    print( remote.updateB( local.getB() ));  // getB, print
}
```

Partitions to:

input.put(“X”, local.getA() );  // getA
input.put(“Y”, local.getB() );  // getB

.... execute updates on server

print( result.get(“A”) );  // print
print( result.get(“B”) );  // print

Compiler Error!
Batch Summary

Client
Batch statement: compiles to Local/Remote/Local
Works in any language (Java, C#, 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
*Forests* (trees) of primitive values (no serialization)
Efficient and portable
Batch Script Language

\[ e ::= x \mid c \]

- \[ e \mid \text{variables, constants} \]
- \[ \text{if e then e else e} \mid \text{conditionals} \]
- \[ \text{for} \oplus x : e \text{ do e} \mid \text{loops} \]
- \[ \text{let} x = e \text{ in e} \mid \text{binding} \]
- \[ x = e \mid e.x = e \mid \text{assignment} \]
- \[ e.x \mid \text{fields} \]
- \[ e.m(e, \ldots, e) \mid \text{method call} \]
- \[ e \oplus \ldots \oplus e \mid \text{primitive operators} \]
- \[ \text{in}_x \mid \text{out}_x e \mid \text{parameters and results} \]
- \[ \text{fun}(x) e \mid \text{functions} \]

\[ \oplus = + - \ast / \% ; \quad < \quad \leq \quad \geq \quad \text{not} \]

Agree on \textit{script format}, not on \textit{object representation}
30 years

SQL

Database Clients

ODBC
JDO/EJB
LINQ
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)

// 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

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:**

```bash
< for (f : *.Files)
    if (f.Size > 90) { outA(f.Name); outB(f.Size) } >
```

**SQL:**

```
SELECT f.Name, f.Size
FROM Files
WHERE f.Size > 90
```
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
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
Web Service Documents are batches
Amazon Web Service

<ItemLookup>
  <AWSAccessKeyId>XYZ</AWSAccessKeyId>
  <Request>
    <ItemIds>
      <ItemIds>
        <ItemIds>1</ItemIds>
        <ItemIds>2</ItemIds>
      </ItemIds>
    </ItemIds>
    <IdType>ASIN</IdType>
    <ResponseGroup>SalesRank</ResponseGroup>
    <ResponseGroup>Images</ResponseGroup>
  </Request>
</ItemLookup>
Amazon Web Service

```xml
<ItemLookup>
  <AWSAccessKeyId>XYZ</AWSAccessKeyId>
  <Request>
    <ItemIds>
      <ItemId>1</ItemId>
      <ItemId>2</ItemId>
    </ItemIds>
    <IdType>ASIN</IdType>
    <ResponseGroup>SalesRank</ResponseGroup>
    <ResponseGroup>Images</ResponseGroup>
  </Request>
</ItemLookup>
```

Custom-defined language for each service operation
Amazon Web Service

```xml
<ItemLookup>
  <AWSAccessKeyId>XYZ</AWSAccessKeyId>
  <Request>
    <ItemIds>
      <ItemId>1</ItemId>
      <ItemId>2</ItemId>
    </ItemIds>
    <IdType>ASIN</IdType>
    <ResponseGroup>SalesRank</ResponseGroup>
    <ResponseGroup>Images</ResponseGroup>
  </Request>
</ItemLookup>

for (item : Items) {
  out_A( item.SalesRank )
  out_B( item.Images )
}
```
Web Service Client Invocation

```java
// create request
ItemLookupRequest request = new ItemLookupRequest();
request.setIdType("ASIN");
request.getItemId().add(1);
request.getItemId().add(2);
request.getResponseGroup().add("SalesRank");
request.getResponseGroup().add("Images");

// execute request
items = amazonService.itemLookup(null, awsAccessKey,
    associateTag, null, null, request, null,
    operationRequest);

// use results
for (item : items.values)
    display( item.SalesRank, item.SmallImage );
```

- Method names in strings
- Batch execution pattern (again!)
Batch Version of Web Service

// calls specified in document

```java
batch (Amazon aws : awsConnection) {
    aws.login("XYZ");

    Item a = aws.getItem("1");
    display( a.SalesRank, a.SmallImage );

    Item b = aws.getItem("2");
    display( b.SalesRank, b.SmallImage );
}
```

Fine-grained logical operations

Coarse-grained execution model
Available Now...

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

**Batch statement as “for”**
```java
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 → Server*   Client → Server → Server
  Client ↔ 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: \textit{reforestation} for bulk data transfer
Transactional, Efficient
(POPL 2007, DBPL 2011)

RPC
Ease of Programming
(ECOOP 2009)

batch

SQL

Web Services
Cross-platform
(ECOWS 2009)

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!