Implementing Remote Procedure Calls

(Slides adapted from an earlier presentation by Abdussalam Alawini)
Overview

1. Introduction
   - Remote Procedure Call vs Local Procedure Call
   - Design Choices

2. RPC Implementation
   - Basic Structure
   - Binding
   - Network Protocol
   - Exception Handling
   - Use of Processes

3. Evaluation
1. Introduction
We’ve studied the:
- Procedural (thread-based) model
- Message passing (event-based) model
- Duality of the two models
- Wrappers to translate calls between models

Here we study an implementation of a procedural model on a distributed, message passing substrate:
- Conceptually, threads pass from one machine to the other and back again
- The programmer is unaware of the underlying message-based substrate
What is a “Remote” Procedure Call?

Communication

Network

Server

Client

\[ Z = F(x, y) \]

\[ F(x, y) \]

result

Compute

\[ F(x, y) \]

Communication Network
Design Alternatives

Explicit message passing
  - i.e. force the programmer to change models

Shared virtual address space
  - Access remote data ... but how to move computation?

Remote fork
  - Parallelism in addition to distribution

Remote procedure call
  - threads libraries, such as Pthreads, make a similar choice
Goals

Simplicity
- Make RPCs as similar to procedure calls as possible
- Simplify distrusted computation

Efficiency
- Anticipated heavy use justifies special support

Security
- Secure end-to-end communications with RPC
2. Implementing RPC
RPC Structure

Caller Machine
- User (application code module)
- User-Stub
- Caller instance of RPCRuntime (RPC communications package)

Callee Machine
- Server (server code module)
- Server-stub
- Callee instance of RPCRuntime
RPC Components Interactions

1. **Caller Machine**
   - User Application
   - User-stub
     - Call Packet
   - Caller RPCRuntime

2. **Network**

3. **Callee Machine**
   - Libraries
     - (Server code)
   - Server-stub
   - Callee RPCRuntime

4. **Exchange of Messages**
   - Call Packet: Pass to server-stub
   - Results Packet: Pass them to server-stub
   - Do work and return results

5. **Process Flow**
   - Import Interface
   - unpack & make local call
   - Do work and return results
   - result to local call
Who Does What?

- **Caller Machine**
  - User Application
  - User-stub
  - Caller RPCRuntime

- **Callee Machine**
  - Libraries (Server code)
  - Server-stub
  - Callee RPCRuntime

- **Mesa Interface Modules**
  - Programmer
  - Lupine (Auto Generation)
  - Part of Cedar

- **Network**
Binding Process

How does a client of the binding mechanism specify what to be bound to?
- Naming

How does the caller specify the callee machine address and the specific procedure to invoke?
- Locating
Naming (Interface Name)

Type: Which interface the caller expects the callee to implement.
- Service Name (e.g. Mail-server)

Instance: Which particular implementer of an abstract interface is desired.
- Machine Address (e.g. Specific mail-server address)
Design Alternatives for Location

1- Include network address in user application
   Binding is too early!

2- Broadcasting Protocol
   Too much interference with innocent bystanders
   Not convenient for binding machines not in the same local network
   Not scalable
Grapevine Database
Locating an appropriate exporter

<table>
<thead>
<tr>
<th>Type (Group)</th>
<th>Member-list</th>
<th>Instance (Individual)</th>
<th>Connect-site</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileAccess</td>
<td>{Ebbets, Luther, Facc}</td>
<td>Ebbets</td>
<td>3#22#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Luther</td>
<td>3#276#</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Facc</td>
<td>3#43#</td>
</tr>
</tbody>
</table>

Export Interface
Type FileAccess
Instance Ebbets

Import Interface
Type FileAccess
Instance Ebbets
Exporting an Interface
(Making it available to a client)

RPCRuntime
Record exported interface in a table

Set the address of the current machine in the connect-site

Adds the instance to the member-list of the type of the instance

Do update

Do update

AddMember

RPCRuntime

Server-stub
Export [FA, Ebbets,...]

Server
Export [FA, Ebbets]

Interface exported (Can be accessed remotely)

Table [tableIndex, InterfaceName, Dispatcher, UniqueID]

SetConnect

AddMember

Return
Binding An Importer to An Exporter Interface
(Getting ready for remote calls)
1- Retrieve binding information
2- Prepare call packet

Alright, I will transmit it to the machine with address “connect-site”

Exporter Interface is ready for your next remote procedure calls

Now let see how this will work

Lookup the table of current exports
Send the corresponding binding info

Call the local procedure using provided arguments

Call user-stub
“I need to import Interface [FA, Ebbets”
Here’s the type and instance we need
OK, I will ask grapevine DB and get the NT address of exporting Interface
Here’s the NT address “connect-site”
Alright, I’ll call the exporter RPCRuntime to get the binding info

Normal call to procedure openFile(y)

OK, now I’ve got UID, TableIndex, and exporter NT Address.
I will record them
Exporter Interface is ready for your next remote procedure calls

1- Retrieve binding information
2- Prepare call packet

Alright, I will transmit it to the machine with address “connect-site”

Lookup current exports
Verify UID
Send call packet to dispatcher
After unpacking, Dispatcher uses the info to map to the right procedure
Call the local procedure using provided arguments

Lookup the table of current exports
Send the corresponding binding info

Call the local procedure using provided arguments
Binding Mechanism Advantages

Stateless: Importing an interface has no effect on the state of the exporting machine.

The use of UID means that bindings are implicitly broken if the exporter crashes and restarts.

Restricting the set of users who can update Grapevine DB.

- To avoid security problems

Several choices of binding time

- Importer specifies the type only
New Transport Protocol (Why?)

Substantial performance gains
Minimize the elapsed real-time between initializing a call and getting results.
   Unacceptable to have large amount of state info.
   Unacceptable to have expensive handshaking.
Guarantee procedure in the server has been invoked precisely once
   Not just “at most once” or “at least once”
Simple Calls Example

Result packet is sufficient Ack to the caller.

User
- Call
  - Send Call Packet
  - Wait Ack or Result
  - Return

RPC+Stub
- Call
  - CallID
  - DP info
  - Seq#

Callee Machine
- [Machine Id, Process]
- Seq#
- [Machine Id, Process]
- Seq#
- CallID
- results
- Send results
- Return

RPC+Stub
- Invoke proc
- Do call

Server
- Send results
- Return
Is this “Stack Ripping”?

In manual stack management the necessary data is taken off the stack and put on the heap.

In RPC it’s taken off the stack and put in a message:
- Sent to the other side
- Put on a different stack ...

Similar to taking the state off the stack and putting it in a continuation:
- Executed by a separate event handler
Complex Call Example
No subsequent call arrived, so ask the caller to do an explicit Ack that it has received the results.

Arguments going to be in 2 packets
Send the rest of the data in Data Pkt
In case of lost packet, long call duration or long gaps between calls retransmit the last pkt and ask for Ack
Satisfying Ack: process waits for results
Caller sends probe pkts periodically (can detect communication failure)
Now all arguments are ready, so the dispatching process starts
Send results
And wait for another call
"works as an Ack"
Return results to caller process (user code)
No subsequent call arrived, so ask the caller to do an explicit Ack that it has received the results
4. Data Integrity and Security
Exception Handling (Remote Process Exception)

Two level of exception

- Communication Failure Exception (Explained with complicated call example), considered to be the primary difference between procedure call and RPC

- Remote Process Exception
If there’s a catch phrase, exception will be handled and results will be sent back to the callee machine.
Use of Processes

RPC now transmit the packet back to the caller (result, exception, or Ack)

If there’s no corresponding process waiting for this packet then it will be dispatched to one of the free processes (Current, New, or Dup)

Part is the process identifiers, part of the activity info within CallID

Process

Network

Caller Machine

Callee Machine

Processes Waiting For RPC packet

<table>
<thead>
<tr>
<th>SrcProcess</th>
<th>DestProcess</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

If there’s no corresponding process waiting for this packet, then it will be dispatched to one of the free processes (Current, New, or Dup).

This is part of the process identifiers, which is part of the activity info within CallID.

RPC now transmits the packet back to the caller (result, exception, or Ack).

If there’s no corresponding process waiting for this packet, then it will be dispatched to one of the free processes (Current, New, or Dup).

Part is the process identifiers, part of the activity info within CallID.

RPC now transmit the packet back to the caller (result, exception, or Ack).
4. Evaluation
Optimizations

Use of thread pool (idle processes) in caller and callee machines to reduce process creation costs.

The use of process source and destination allow processes to get the packets they’re waiting for directly from the interrupt handler.

Use of subsequent packet for implicit acknowledgments of previous packets.

Avoid the cost of establishing and terminating connections by the implementation of packet-level protocol.
Performance Evaluation

They’ve measured the elapsed time between two machines for 12,000 calls for each of the following procedures:

- 0-10 arguments/results.
- 1-100 word array.
- Caller Resume and unwind exception handling.
### Performance Results

Table I. Performance Results for Some Examples of Remote Calls

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Minimum</th>
<th>Median</th>
<th>Transmission</th>
<th>Local-only</th>
</tr>
</thead>
<tbody>
<tr>
<td>no args/results</td>
<td>1059</td>
<td>1097</td>
<td>131</td>
<td>9</td>
</tr>
<tr>
<td>1 arg/result</td>
<td>1070</td>
<td>1105</td>
<td>142</td>
<td>10</td>
</tr>
<tr>
<td>2 args/results</td>
<td>1077</td>
<td>1127</td>
<td>152</td>
<td>11</td>
</tr>
<tr>
<td>4 args/results</td>
<td>1115</td>
<td>1171</td>
<td>174</td>
<td>12</td>
</tr>
<tr>
<td>10 args/results</td>
<td>1222</td>
<td>1278</td>
<td>239</td>
<td>17</td>
</tr>
<tr>
<td>1 word array</td>
<td>1069</td>
<td>1111</td>
<td>131</td>
<td>10</td>
</tr>
<tr>
<td>4 word array</td>
<td>1106</td>
<td>1153</td>
<td>174</td>
<td>13</td>
</tr>
<tr>
<td>10 word array</td>
<td>1214</td>
<td>1250</td>
<td>239</td>
<td>16</td>
</tr>
<tr>
<td>40 word array</td>
<td>1643</td>
<td>1695</td>
<td>566</td>
<td>51</td>
</tr>
<tr>
<td>100 word array</td>
<td>2915</td>
<td>2926</td>
<td>1219</td>
<td>98</td>
</tr>
<tr>
<td>resume except’n</td>
<td>2555</td>
<td>2637</td>
<td>284</td>
<td>134</td>
</tr>
<tr>
<td>unwind except’n</td>
<td>3374</td>
<td>3467</td>
<td>284</td>
<td>196</td>
</tr>
</tbody>
</table>
Performance Summary

For transferring large amounts of data in one direction products other than RPC have advantage.

- Transfer fewer packets in the other direction

They haven’t measure the performance of exporting or importing interfaces.

Used by:

- Alpine [File server supports multi-machine transactions]
- Control communication for an Ethernet-based for telephone and audio project
- Networking games.
Conclusion

RPC is used to communicate between processes in different address spaces
- May be on separate machines or on the same machine

Communication is implicit
- RPC makes distributed programming easier
- Allows thread-based programming model on distributed platforms
- Reliably transfers data and control