Remote Procedure Call and Remote Object Messaging

Class 4
In the beginning…

there were messages

• but the messages were without form!
  – just a string of bytes
  ⇒ develop marshalling libraries
    - int2bytes(anInt) or writeInt(anInt, aStreamOrBuffer)
    - bytesToInt(aByteArray) or readInt(aStream)

• How do I know to unmarshall an int, not a string?
On the second day…

John White invented RPC (1976)
- Implemented efficiently at PARC (1982)

Idea:
- **Procedure Call** is well understood way of transferring data and control within a single computer
- extend it to 2 computers on a network
Goals

programs relatively easy. The existing communication mechanisms appeared to
be a major factor constraining further development of distributed computing.
Our hope is that by providing communication with almost as much ease as local
procedure calls, people will be encouraged to build and experiment with distrib-
uted applications. **RPC will, we hope, remove unnecessary difficulties, leaving
only the fundamental difficulties of building distributed systems: timing, inde-
dependent failure of components, and the coexistence of independent execution
environments.**

We had two secondary aims that we hoped would support our purpose. We
wanted to make RPC communication highly efficient (within, say, a factor of

Goals

• “Make distributed computing easy”
  – By making communication as easy as a local procedure call, they hoped to encourage the writing of distributed applications

• RPC “removes unnecessary difficulties”, leaving only the “fundamental difficulties”
  – timing
  – independent failure
  – coexistence of independent execution environments
Basic Architecture

CLIENT CODE

CLIENT STUB

SESSION LEVEL PROTOCOL

SERVER CODE

SERVER STUB

SESSION LEVEL PROTOCOL

NETWORK

call

return

marshal

unmarshal

demux

execute service

local call

reply

marshal

unmarshal
Principle

The semantics of a remote call should be as close as possible to those of a local call

• Except:
  – You have to name the destination (binding)
  – Sharing of parameters is not possible
  – Independent failures
  – 3rd party references

• What works?
  –
What about Objects?

- *Coulouris et al.* claim that the Object Model is just right for distributed computing

**Object model:**

1. ubiquitous object reference mechanism
2. send messages to objects, with objects as arguments
3. objects respond autonomously by executing *method*
4. objects export an interface
5. state of an object is somewhat encapsulated
6. objects are widely shared
7. objects are *not* explicitly deallocated
On the third day…

came Remote Object Messaging (RMI)

• send an invocation message to a (possibly) remote object

• the identity of that object solves the binding problem

• life is good!
Figure 5.3  Remote and local method invocations

remote invocation

A

remote invocation

B

clocal invocation

clocal invocation

C

clocal invocation

D

remote invocation

E

remote invocation

F
How good is the object model really?

• Object model:

  1. ubiquitous object reference mechanism
     - In a DS, this means that every object must have a global name!
     - Conceptually clean, but expensive to implement
     - Ingalls: the important thing about cheating is not to be caught (in implementing systems, *not* when doing homework!)
       - or at least, all objects must have the potential for a global name
       - cons up a global name only when it is needed
2. send messages to objects, with objects as arguments
   - arguments can’t *always* be object references
   - send copies of an object?
   - what are the consequences
3. objects respond autonomously by executing a *method*
   - this is a great match for distributed systems
   - different objects at different locations can execute different code
4. objects export an interface
   - this is a great match too
5. state of an object is somewhat encapsulated
   - In a DS, state is *really* encapsulated
   - *Object* encapsulation, not class encapsulation
   - no “friends”

6. objects are widely shared
   - In a distributed system, a message to a remote object is 1000 times slower than a message to a local object
   - what impact does this have on wide sharing
   - what impact does partial failure have on sharing?

7. objects are *not* explicitly deallocated
   - but global GC is hard (but memory is cheap)
What’s Important in Distributed Systems?

- Caching and copying as alternatives to remote access
- Immutable objects are a secret weapon
  - Which object models support them?
- Separating failures from exceptions
  - An exception is a result that falls within the specification of the object
  - A failure occurs when an object fails to meet its specification
The RPC Protocol

• Birrell & Nelson argue that using reliable streams for RPC is unacceptable
  – high set-up cost for each RPC (latency)
  – cost of maintaining state for each client
  – stream protocol does more than is required for the particular case of an RPC
  – since payload may be small, overhead is large

• Hence, they developed a special-purpose transport
Goals of PRC Transport

• minimize server load imposed per client
• “exactly once” semantics:
  – if the call returns, the procedure executed once
  – if there is no return, then a failure is indicated
    - procedure may have executed once, or not at all
  – client will wait indefinitely provided server has not crashed
• Efficient when all data will fit in a packet
  – common case is that packet will not be lost
Simple Calls

- One request pkt and one response pkt

Fig. 3. The packets transmitted during a simple call.

- Lost pkts?
- Slow server?
- Slow clients?
Features of the Protocol

- **CallID**
  1. Allows callee to eliminate duplicate requests
  2. Allows caller to match-up responses with requests

- **Threading**
  - No thread can have more than one call outstanding

- **Required state:**
  - Single counter on each client (what about reboots?)
  - “High water mark” CallID per client on the server
  - can eventually be discarded
Complicated Calls

• Transmitter responsible for retransmission
  – retransmitted request asks for explicit ack.
  – handles lost pkts, long calls, and long gaps
• If caller receives *ack* but no *response*
  – sends *probe* packet, which demands an ack
  – why?
• Caller will wait indefinitely so long as probes are ack’d
• Burden of this work is on client, not server
Fig. 4. A complicated call. The arguments occupy two packets. The call duration is long enough to require retransmission of the last argument packet requesting an acknowledgment, and the result packet is retransmitted requesting an acknowledgment because no subsequent call arrived.
### Performance

<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>

- all times in microseconds (µs)
- measured 12 000 calls in each case
- transmission times are calculated, not measured
Threading

- Client needs to be multi-threaded if it needs to continue working while waiting for a reply
  - e.g., to be responsive to the UI
• Server needs to be multi-threaded if
  – Responding to calls is not CPU intensive
  – There is a desire to maximise throughput or minimize latency

![Diagram showing different scenarios of server and client interactions.](image-url)