Remote Procedure Calls (RPC)

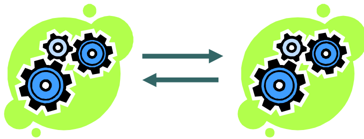
Xin Zheng – CS 614 Fall '07
(Some slides borrowed from Fall '05)

Outline

- What's RPC
- Cedar and Firefly RPC designs
- Performance measurements
- Conclusion

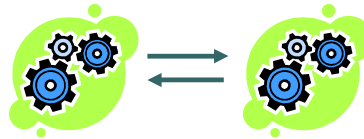
Network/Distributed Programs

- Computers/processes want to talk to each other
- Might be different machines in a network, or different address spaces in same machine



Communication

- Can design your own protocol



Custom Protocol

```
public class Message {
  public Message(String type,
                 String args) {…}
}
```

Custom Protocol

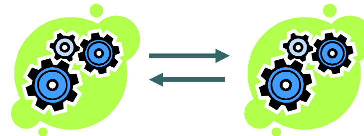
```
public class Communicator {
  public void send(Message m) {…}
  public Message recv() {…}
}
```

Custom Protocol

```
public void joinGame(String gameName) {
  Message m = join(gameName, playerName);
  game_comm.send(m);
  Message reply = game_comm.recv();
  if (reply.getType() == SUCCESS) {
    …
  } else if (reply.getType() == ERROR) {
    …
  } else {…}
}
```

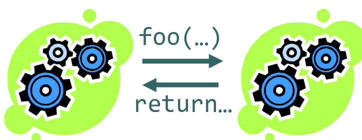
Communication

- Can design your own protocol
- OR…
- Realize that intra-process communication already happens all the time, via procedure calls!



A Very Simple Idea

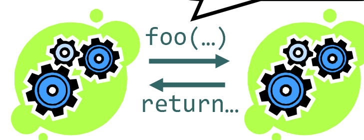
- Retain local procedure call semantics, but let procedures reside on different machines
- …And you get RPC!



A Very Simple Idea

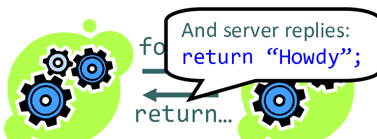
- Retain local procedure call semantics, but let procedures reside on different machines
- …And you get RPC!

Client wants to say something:
`foo("Hi there");`



A Very Simple Idea

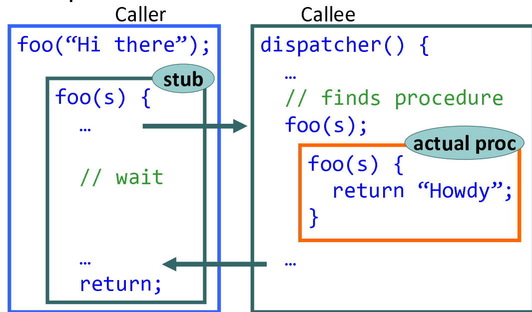
- Retain local procedure call semantics, but let procedures reside on different machines
- …And you get RPC!



Why RPC?

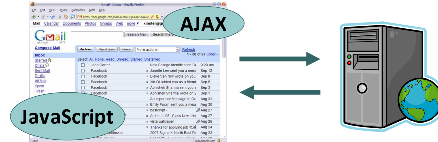
- For programmers, nothing new to learn
- Distributed applications don't have to look all that different from local programs
- Reducing extralinguistic clutter is always good…. Pretty much all languages already support procedures/functions/methods

How It Works



Web Applications

- Server and client (browser) side components
- Client provides UI
- Server provides data, computation




```
// send_to.onchange event handler
function findContacts() {
  a = document.forms[0].send_to.value;
  s =
  con
```

```
<input type="text"
 name="send_to"
 onchange="findContacts()">
```

AJAX = RPC!

Google Web Toolkit

- Java to JavaScript compiler
- Most of Gmail, Google Maps, etc written in GWT
- Provides higher level abstractions in Java compared to native JavaScript

Google Web Toolkit

```
public interface ContactsService {
  public String[] getContacts(String s);
}
```

Google Web Toolkit

```
public interface ContactsService { …
```

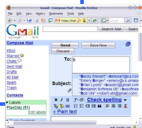
```
public class ContactsServiceImpl
  implements ContactsService {

  public String[]
        getContacts(String s) {
    return …;
  }
}
```

Google Web Toolkit

```
public interface ContactsService { …
```

```
// send_to.onchange event handler
public void onChange(String s) {
  ContactsService svc =
    GWT.create(ContactsService);
  String[] contacts =
    svc.getContacts(s);
  …
}
```



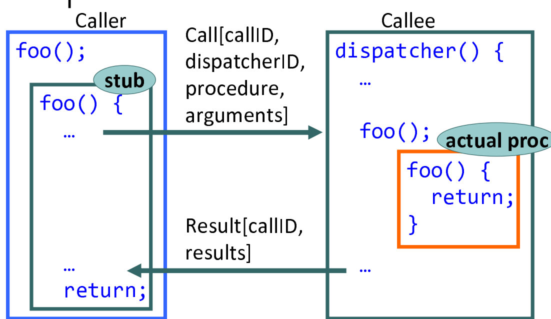
Making RPC Fast

- Both Cedar and Firefly use custom protocols
- Skips traditional network stack
- Very platform specific
  - Firefly has some (a lot?) hand-tuned assembly code
  - Also relies a bit on multi-processors for performance

Cedar

- Minimize time between call and getting result
- Minimize load on servers
- Assume a large number of call with small amounts of data transfer
- Protocol defined at the packet level
- Implemented in Mesa

Simple Calls



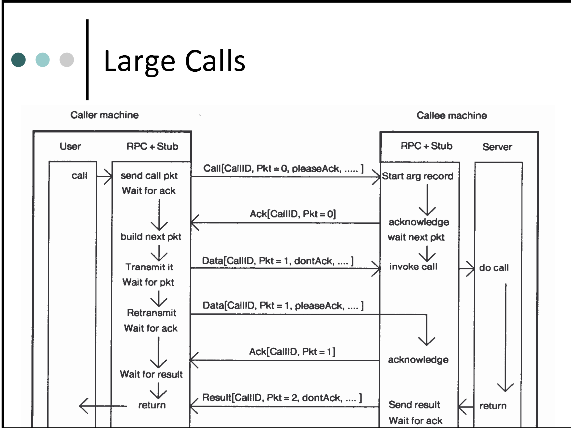
Simple Calls

- Client retransmits until ack received
  - Result acts as an ack
  - Similar for the callee: next call packet is a sufficient ack)
- Callee maintains table for last call ID
  - Duplicate call packets can be discarded
  - This shared state acts as connection

Advantages

- No special connection establishment
- Low state requirements
  - Callee: only call ID table stored
  - Caller: single counter sufficient (for sequence number)
  - No concern for state of connection – ping packets not required
  - No explicit connection termination

Large Calls



Firefly

- Goes even further than Cedar in optimizations
- Protocol defined at the sub-packet level
  - Allows procedure stubs to access packets directly
  - Packet are reused when possible
- Written in Modula-2+/assembly

Reusing Packets

- Server stub can retain call packet for result
- Waiting thread contains packet buffer – this packet can be used for retransmission
- Packet buffers reside in memory shared by everyone
  - Security can be an issue

Performance of Cedar

Table I. Performance Results for Some Examples of Remote Calls

Procedure	Minimum	Median	Transmission	Local-only
no args/results	1059	1097	131	9
1 arg/result	1070	1105	142	10
2 args/results	1077	[illegible]	[illegible]	11
4 args/results	1115	[illegible]	[illegible]	12
10 args/results	1222	[illegible]	[illegible]	17
1 word array	1069	1111	[illegible]	10
4 word array	1106	1153	174	13
10 word array	1214	1250	239	16
40 word array	1643	1695	566	51
100 word array	2915	2926	1219	98
resume except'n	2555	2637	284	134
unwind except'n	3374	3467	284	196

≈ 1.1 ms per call to "Null"

Performance of Firefly

Table I: Time for 10000 RPCs

# of caller threads	Calls to Null()		Calls to MaxResult(b)	
	seconds	RPCs/sec	seconds	megabits/sec
1	26.61	375	63.47	1.82
2	16.80	595	35.28	3.28
3	[illegible]	[illegible]	27.28	4.25
4	[illegible]	[illegible]	24.93	4.65
5	[illegible]	[illegible]	24.69	4.69
6	[illegible]	[illegible]	24.65	4.70
7	13.49	741	24.72	4.69
8	13.67	732	24.68	4.69

≈ 2.7 ms per call to "Null"

Firefly Stubs+Runtime

Table VII: Latency of stubs and RPC runtime

Machine	Procedure	Microseconds
Caller	Calling program (loop to repeat call)	16
	Calling stub (call & return)	90
	Starter	128
	Transporter (send call pkt)	27
Server	Receiver (receive call pkt)	158
	Server stub (call & return)	68
	Null (the server procedure)	10
	Receiver (send result pkt)	27
Caller	Transporter (receive result pkt)	49
	Ender	33
	TOTAL	606

Send+ Receive

Firefly Send+Receive

Table VI: Latency of steps in the send+receive operation

Action	Microseconds for 74 byte packet		Microseconds for 1514 byte packet (if different)	
F[illegible]	59	a		
[illegible]	45	b	440	b
[illegible]	37	a		
[illegible]	39	a		
[illegible]	10	c		
[illegible]	76	a		
[illegible]	22	a		
[illegible]	70	d	815	e
[illegible]	60	d	1230	e
[illegible]	80	d	835	e
[illegible]	14	a		
Ha[illegible]	177	a		
Calc[illegible] UDP checksum	45	b	440	b
Wakeup RPC thread	220	a		
Total for send+receive	954		4414	

Replace unneeded buffers and process outstanding packets

Assembly Code

Table IX: Execution time for main path of the Ethernet interrupt routine

Version	Time in microseconds
Original Modula-2+	758
Final Modula-2+	547
Assembly language	177

Processors

Table X: Calls to Null() with varying numbers of processors

caller processors	server processors	seconds for 1000 calls
5	5	2.69
4	5	2.73
3	5	2.85
2	5	2.98
1	5	3.96
1	4	3.98
1	3	4.13
1	2	4.21
1	1	4.81

Threads

Table XI: Throughput in megabits/second of MaxResult(b) with varying numbers of processors

caller processors server processors	5 5	1 5	1 1
1 caller thread	2.0	1.5	1.3
2 caller threads	3.4	2.3	2.0
3 caller threads	4.6	2.7	2.4
4 caller threads	4.7	2.7	2.5
5 caller threads	4.7	2.7	2.5

Comparison

Table XII: Performance of remote RPC in other systems

System	Machine - Processor	~ MIPs	Latency in milliseconds	Throughput in megabits/sec
Cedar [2]	Dorado - custom	1 x 4	1.1	2.0
Amoeba [7]	Tadpole - M68020	1 x 1.5	1.4	5.3
V [3]	Sun 3/75 - M68020	1 x 2	2.5	4.4
Sprite [6]	Sun 3/75 - M68020	1 x 2	2.8	5.6
Amoeba/Unix [7]	Sun 3/50 - M68020	1 x 1.5	7.0	1.8
Firefly	FF - MicroVAX II	1 x 1	4.8	2.5
Firefly	FF - MicroVAX II	5 x 1	2.7	4.6

Comments

- RPC, as an abstraction, is popular
  - Both inter- and intra-machine
- Asynchronous versions now common
- Tension between interoperability and performance
  - Java RMI's default implementation is HTTP
  - As is GWT