Lightweight Remote Procedure Call

Brian N Bershad, Thomas E Anderson, Edward D Lazowska and Hendry M Levy

Presented by Matthaus Litteken
Why lightweight?
Is RPC too heavy?
Not for Remote Calls
So what's the problem?
Most RPCs are made between subsystems on the same machine

V (97%)

Taos (94.7%)

Sun UNIX & NFS (99.4%)
That's not very remote
But what's the problem?
Overhead
How bad could it be?
### NULL Cross-Domain Call

<table>
<thead>
<tr>
<th>System</th>
<th>Minimum</th>
<th>Actual</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accent</td>
<td>444</td>
<td>2300</td>
<td>1856</td>
</tr>
<tr>
<td>Taos</td>
<td>109</td>
<td>464</td>
<td>355</td>
</tr>
<tr>
<td>Mach</td>
<td>90</td>
<td>754</td>
<td>664</td>
</tr>
<tr>
<td>V</td>
<td>170</td>
<td>730</td>
<td>560</td>
</tr>
<tr>
<td>Amoeba</td>
<td>170</td>
<td>800</td>
<td>630</td>
</tr>
<tr>
<td>DASH</td>
<td>170</td>
<td>1590</td>
<td>1420</td>
</tr>
</tbody>
</table>
Okay, so where did all that overhead come from?
Stub overhead
Message buffer overhead
Access validation
Message transfer
Scheduling
Context switch
Dispatch
So what's the solution?
Merge subsystems

Why bother using RPC if the systems can be merged?

It reduces security and safety
So what's the solution?
How does it work?
Server exports an interface

Clients bind to those interfaces

Clients call using the client-stub

Servers handle the call using server-stub

Server returns to client-stub
Okay, so how is that different?
Okay, so how is that different?

Clients make an import call to kernel
  Kernel notifies server's clerk
Server has exported interfaces to the kernel via an LRPC clerk
  The clerk hands off a PDL to the kernel
    PDL contains one PD per procedure in interface
      Kernel allocates shared A-stack for server and client
        Kernel returns a Binding Object and A-stack list to client
Now the client stub does some work

Client-stub grabs an A-stack from its list
Pushes procedure's arguments onto the A-stack
Stores A-stack address, Binding Object, procedure identifier into registers
Traps to kernel

Now the kernel does some work
Now the kernel does some work

Validates the call
Records return address, stack pointer into A-stack linkage
Pushes linkage onto linkage stack
Updates client thread to run off new E-stack in server domain
Loads virtual memory registers with server domain entries
Upcalls into the server stub to begin execution

Now the server stub does some work
Now the server stub does some work

Server stub branches directly into server procedure
  Server has direct access to client's A-stack
    Procedure code runs
    Procedure returns via server stub
  Server stub traps to kernel
    Kernel returns control to the thread
Okay, so how is that better?

Lower Overhead
Lower Overhead

Shared stack between client and server
One call, two returns
Kernel only validates on call, not return
Less scheduling overhead
Less copying, fewer messages
Fewer context switches
Okay, so how much better?

Three times as fast as SRC RPC on single processor

Up to five times as fast on four processors

SRC RPC plateaus at two processors (coarse locking)
Wait, how does it work on multi-processors?

Idle processors cache domain contexts

On a call, kernel checks for idle processor with the right domain cached
Kernel moves calling thread onto idle processor, execution continues without context switch
Idle thread continues idling, but on the original client processor
Single lock for every A-stack list/queue
Okay, but what if the call actually is remote?

First stub instruction determines if the call is cross domain or cross machine

Bit is set in Binding Object

Execution branches to a conventional RPC stub
Okay, but what about exceptions?
Server Side

Binding Object gets revoked
Thread gets restarted in client with call-failed exception
Client handles exception

Client Side

If there is an outstanding call, it can't return to crashed client code
Lightweight Remote Procedure Calls

Best when the call isn't Remote
Still as good when it is Remote
Reduces the standard RPC in many ways
 Doesn't sacrifice security for speed
Outperforms by far on multi-processors

Great, so why use regular RPC?
That's all!

Thanks for listening!

http://cs533.herokuapp.com