Remote Procedure Calls

Matt Mukerjee

Why RPC?

- Clean, familiar semantics
  - Distributed Systems are hard to begin with!
- Efficient (?)
- Generality
  - parallels single machine functional decomposition

- Make the programmer's life easy!

Implementing RPC

- Andrew Birrell
  - Xerox PARC, then DEC SRC
  - DEC SRC responsible for Firefly workstation
    - used in Bershad paper
  - now at Microsoft Research
- Bruce Nelson
  - Xerox PARC, then Cisco
  - CMU PhD – thesis the “foundation” of RPC
  - ACM Software Systems award (for RPC)

RPC – Take Home Points

- Treat cross-machine calls like local calls
- Let's make the programmer's life easy

- New Failure conditions
  - think Brewer's Conjecture

Overview

- RPC Structure
  - Functions
  - Stubs
  - RPCRuntime
- RPC Implementation
  - Binding
  - Transport Protocol
- RPC Evaluation
- Issues

Reexamine Local Procedure Calls

- A calls B
- A waits for B
- B does the work, returns control to A
- A resumes

Applied to RPC

- A calls B *on a different machine*
- A waits for B, *other processes run*
- B does the work, *sends a message* to A
- A resumes

Stubs

- Stubs provide:
  - entry-point into remote functions
  - functional prototypes
- Stubs automatically generated

RPCRuntime

- Handles:
  - retransmissions
  - acknowledgments
  - packet routing
  - encryption

Simple Call

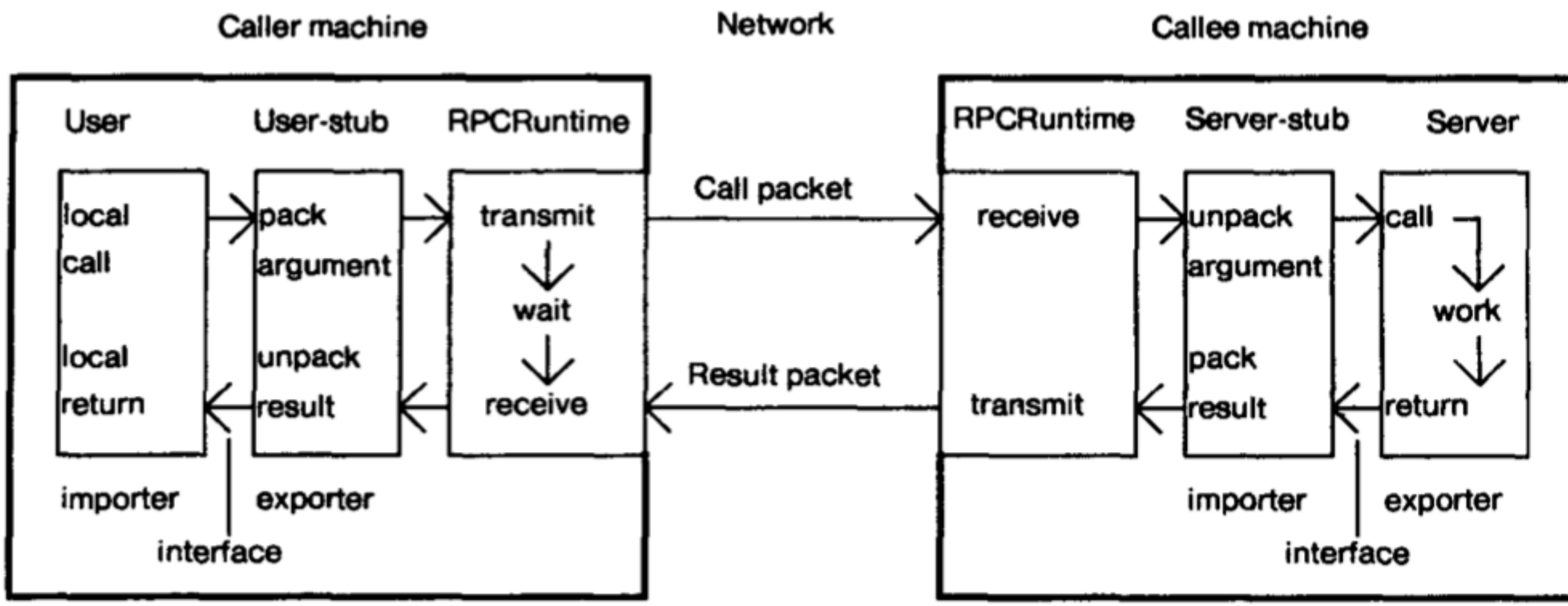

Fig. 1. The components of the system, and their interactions for a simple call.

Overview

- RPC Structure
  - Functions
  - Stubs
  - RPCRuntime
- RPC Implementation
  - Binding
  - Transport Protocol
- RPC Evaluation
- Issues

Binding

- Uses types and instances:
  - Type: mail server
  - Instance: mail.website.com
- Uses “Grapevine” as a lookup server
  - Similar to DNS
- Can bind by:
  - network address
  - instance name
  - type name

RPC Transport Protocol - Requirements

- RPC mainly short messages between machines
  - Latency is important
  - Small packets with low overhead is ideal
- RPC must always fail or execute exactly once
- Best case:
  - Caller sends a *call packet* to server
  - Server does the work
  - sends back a *result packet*

RPC Transport Protocol – Potential Issues

- If the server takes to long to respond:
  - it could be packet loss!
  - duplicate packets
  - *Call identifier* silently drop duplicate packets
  - But…both machines must maintain state info
- Multi-packet argument case:
  - Clever acknowledgement system to reduce traffic
  - But…bad at sending bulk data

Overview

- RPC Structure
  - Functions
  - Stubs
  - RPCRuntime
- RPC Implementation
  - Binding
  - Transport Protocol
- RPC Evaluation
- Issues

Evaluation

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	1127	152	11
4 args/results	1115	1171	174	12
10 args/results	1222	1278	239	17
1 word array	1069	1111	131	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

Possible Issues

- Why do some people dislike RPC?
- Machine/communication failure
- Overhead from lack of shared address space
- Data integrity/security

- Grapevine server could fail
- DNS-like attack on Grapevine

Strengths and Weaknesses

- It’s “humble”:
  - “There are certain circumstances in which RPC seems to be the wrong communication paradigm”

- Other works not referenced, just alluded to
- Benchmarks not meaningful

Where did RPC Go?

- Hot topic in the 80's / 90's
- All but disappeared?

- Sockets, etc. caught up…
- Moore's law made it irrelevant
  - (M. Satyanarayanan – Coda paper)

Lightweight Remote Procedure Call

- Brian Bershad
  - UW PhD, wrote SPIN, now a professor at UW
- Thomas Anderson
  - UW PhD, tons of papers, also professor at UW
- Edward Lazowska
  - UW Professor
- Hank Levy
  - UW Professor, part of the DEC VAX design team

LRPC – Take Home Points

- RPC was pervasive
  - Remote calls
  - Local calls across "protection domains"
  - Simple calls with few parameters
- Local communication much more frequent
  - Optimize it

- Optimize the common case!
- Treat the uncommon case differently

LPRC Motivation

- Local RPC had awful performance
  - Programmers coded around it
- LRPC is much faster
  - Programmers to design better code
- Monolithic kernels have no intra-OS processes boundaries
  - Not secure!
  - Makes it hard to debug, modify, etc.

Overview

- LRPC Structure
- LRPC Implementation
  - Domain Caching
- LRPC Evaluation
- Wrap-up

LRPC Structure

- Almost identical to RPC except for the focus on:
  - Keeping logically separate part separate
    - RPC does this…by having them on different machines
  - Keeping control transfer and stubs simple
  - Sharing VM (parameters) between client and server
  - Using concurrency
- Must keep overhead low in the common case!

LRPC Implementation

- Many “cute” hacks for speed:
  - Clients pass data to servers through VM mapping
  - Procedures in same interface can share “argument stacks”
  - Keeps “execution stacks” available in server domain
  - Uses “domain caching” on multiprocessor machines

Multiprocessor LRPC

- TLB misses (from context switching) are expensive, so they use *domain caching*:
  - Eg: Processor A is idling in kernel-space
  - Processor B makes LRPC call from user-space to kernel-space
  - Instead of running in kernel-space on Processor B, the function runs on Processor A
- This means no context switch!

Other Benefits of LRPC

- Less argument copying needed
- Private channel between domains
- In cases where parameters are immutable even less copies can be achieved

Overview

- LRPC Structure
- LRPC Implementation
  - Domain Caching
- LRPC Evaluation
- Wrap-up

Evaluation

Table IV. LRPC Performance of Four Tests (in microseconds)

Test	Description	LRPC/MP	LRPC	Taos
Null	The Null cross-domain call	125	157	464
Add	A procedure taking two 4-byte arguments and returning one 4-byte argument	130	164	480
BigIn	A procedure taking one 200-byte argument	173	192	539
BigInOut	A procedure taking and returning one 200-byte argument	219	227	636

Evaluation

- 3 times faster than the built-in RPC
  - Not an order of magnitude difference
- Gets much closer to the theoretical minimal

- Multiprocessor version close to throughput cap
- Multiprocessor version is scalable

Strengths and Weaknesses

- Simple, cute hacks, better than optimized version
- Comes up with secondary ideas
  - Domain caching

- Didn't try to port their code to other architectures
  - "[it] should be a straightforward task"
- Argument stacks in global shared virtual memory
  - Doesn't match design specifications
  - Lowered security

Performance of Firefly RPC

- Basically response of the Firefly team to LRPC
- Cute hacks for the remote machine case
  - LRPC covered the local machine case

RPC in the *x*-Kernel

- Clever idea:
  - RPC-like system, change protocols' layers at runtime
  - Change the underlying network layer (from IP to direct-to-Ethernet) at runtime

Discussion