A Brief History of Distributed Programming: RPC

YOW Brisbane 2016
Caitie McCaffrey
Distributed Systems Engineer

@caitie
caitiem.com
Concurrent Programming

Computation can advance without waiting for all other computations to complete

Distributed Programming

Computation is accomplished via the communication and coordination of networked computers
“An asynchronous event driven JavaScript runtime”

Single Threaded
Event Loop
Callbacks
Golang

Go Routines

Channels
How do you communicate with code running on other machines?
How do you communicate with code running on other machines?

We’ve left that as an exercise for the reader...
Erlang

Actor Model for Concurrency

Transparent Distribution Added Later
Cloud Haskell

Haskell Implementations of Distributed Erlang semantics

Extension of a Concurrent Language for Distribution
RPC: Remote Procedure Call

make remote calls just as simple as local calls
Overview

1974
RFC 674

1975
RFC 684

1976
RFC 707

1984
Implementing Remote Procedure Calls

1988
A Critique of the Remote Procedure Call

1989
RFC 1094
NFS

1991
CORBA

1994
A Note on Distributed Computing

Present
Future
Paradigm (PCP) is an attempt at defining a mechanism for resources sharing across all 70 nodes on the Internet.

Procedure Call Protocol Documents
Version 2

As many of you may know SRI is part of a team working on the National Software Works project. In the course of our work we have developed a Procedure Call Protocol to be used between the modules which make up the NSW. We are interested in your comments on this protocol. Please forward your remarks to either:

James E. White (WHITE@SRI-ARC) or Jon Postel (POSTEL@SRI-ARC)

Augmentation Research Center
Stanford Research Institute
Menlo Park, California 94025

(415) 326-6200 x2960 (White) or x3718 (Postel)

This note announces the release of the second published version of several National Software Works (NSW) and Procedure Call Protocol (PCP) documents. Version 2 is SUBSTANTIALLY different than Version 1; it and all intermediate, informally distributed PCP documents are obsoleted by this release.

Each of the following documents is available on-line in two forms: as an NLS file and as a formatted text file. The Journal number (e.g. 24459) refers to the former, of course, and the pathname (e.g. [SRI-ARC]<NLS>PCP.TXT) to the latter, accessible via FTP using USER=ANONYMOUS and PASSWORD=GUEST (no account required). Let it be emphasised that files indicated by pathname of the form
RFC 684 Criticisms

Local Calls & Remote Calls have different Cost Profiles

Remote Calls can be delayed or never return

Asynchronous Message Passing is a better model because it makes the passing of messages explicit
Generalization to functions

Generalize TELNET and FTP’s call-and-response model to functions from an application-specific grammar. One port for all protocols.

Control Flow Critique

RPC only allows for sequential composition.
Implementing Remote Procedure Calls

First Commercial Grade RPC

Fig. 3. The packets transmitted during a simple call.
A Critique of the Remote Procedure Call Paradigm

Andrew S. Tanenbaum
Robbert van Renesse†
Dept. of Mathematics and Computer Science
Vrije Universiteit
Postbus 7161
1007 MC Amsterdam, The Netherlands

The remote procedure call paradigm is widely used in distributed operating systems. It is conceptually simple to use and straightforward to implement. Nevertheless, experience has shown that it also has some subtle, but less pleasant aspects. In this paper we discuss problems with RPC in the areas of conceptual problems with the model itself, technical problems with implementing it, problems caused by client and server crashes, problems caused by heterogeneous systems, and performance problems. The paper concludes with a discussion and analysis of the problems and proposed solutions.

1. INTRODUCTION

Within the operating system research community, remote procedure call [Birrell and Nelson, 1984; Nelson, 1981] has achieved sacred cow status. It is almost universally assumed to be the appropriate paradigm for building a distributed operating system. Through our use of remote procedure call (RPC) in our own experimental distributed system [references to be provided after blind refereeing is completed], we have discovered that although RPC is an elegant model, it also has a number of unpleasant aspects as well. In this paper we have assembled some of our criticisms, as well as those of other researchers, not because we believe RPC should be abandoned, but as a way to focus attention on the problems and to stimulate others to try and solve them.

Before detailing our criticisms of the RPC model, let us briefly summarize what we mean by RPC. RPC is a communication mechanism between two parties, a client and a server. For simplicity, let us assume that a computation consists of a main program, running on the client machine, and a procedure to be called, running on the server machine.

When the main program calls the procedure, what actually happens is that a call is made to a special procedure called the client stub on the client’s machine. The client stub performs several tasks, including marshaling the arguments to the procedure, sending the call to the server machine, and handling the return value and error conditions. The client stub also checks the calling arguments for validity and legality, and performs error handling.

“...
“Imagine that two programmers are working on a project. Programmer 1 is writing the main program. Programmer 2 is writing a collection of procedures to be called by the main program”
“At the very last minute, after all the code has been thoroughly tested, debugged, and documented and both programmers have quit their jobs and left the country ...”

A Critique of the Remote Procedure Call Paradigm
“...the project management is forced by unexpected, external circumstances to run the program on a distributed system.”

A Critique of the Remote Procedure Call Paradigm
“...the project management is forced by unexpected, external circumstances to run the program on a distributed system.”

A Critique of the Remote Procedure Call Paradigm
“It is our contention that a large number of things may now go wrong due to the fact that RPC tries to make remote procedure calls look exactly like local ones, but is unable to do it perfectly”

A Critique of the Remote Procedure Call Paradigm
“There is, in fact, no protocol that guarantees that both sides definitely and unambiguously know that the RPC is over in the face of a lossy network.”

A Critique of the Remote Procedure Call Paradigm
This RFC describes a protocol that Sun Microsystems, Inc., and others are using. A new version of the protocol is under development, but others may benefit from the descriptions of the current protocol, and discussion of some of the design issues. Distribution of this memo is unlimited.

1. INTRODUCTION

The Sun Network Filesystem (NFS) protocol provides transparent remote access to shared files across networks. The NFS protocol is designed to be portable across different machines, operating systems, network architectures, and transport protocols. This portability is achieved through the use of Remote Procedure Call (RPC) primitives built on top of an External Data Representation (XDR). Implementations already exist for a variety of machines, from personal computers to supercomputers.

The supporting mount protocol allows the server to hand out remote access privileges to a restricted set of clients. It performs the operating system-specific functions that allow, for example, to attach remote directory trees to some local file system.

1.1. Remote Procedure Call

Sun's Remote Procedure Call specification provides a procedure-oriented interface to remote services. Each server supplies a "program" that is a set of procedures. NFS is one such program. The combination of host address, program number, and procedure number specifies one remote procedure. A goal of NFS was to not require any specific level of reliability from its lower levels, so it could potentially be used on many underlying transport protocols, or even another remote procedure call implementation. For ease of discussion, the rest of this document will assume NFS is implemented on top of Sun RPC, described in RFC 1052, "RPC: Remote Procedure Call Protocol Specification".

1.2. External Data Representation

The eXternal Data Representation (XDR) standard provides a common way of representing a set of data types over a network. The NFS Protocol...
Network File System

**Soft Mounting**
- Introduced new error codes for distributed failures that existing UNIX applications could not handle

**Hard Mounting**
- Operations would block until they could be completed successfully
Supported Cross-Language, Cross Address Space Interoperability for Object-Oriented Programming

Interface Definition Language (IDL) : used to generate stubs for remote objects & mappings between different primitive types

“It’s Just a Mapping Problem” Remote to local exception mapping, remote to local method invocation.
“It is the thesis of this note that this unified view of objects is mistaken”
A Note on Distributed Computing

Latency
Performance analysis is non-trivial and one design is not always going to be the right design.

Memory Access
How do we deal with the problems of pointers and references? Once moved they are no longer valid unless we use distributed shared memory.

Partial Failure
Failures are detectable in the local environment and result in a “return of control”. In distributed computing this isn't true.
Two Paths Forward

Treat all objects as local

Or

Treat all objects as remote

A Note on Distributed Computing
“This approach would also defeat the overall purpose of unifying the object models. The real reason for attempting such a unification is to make distributed computing more like local computing and thus make distributed computing easier. This second approach to unifying the models makes local computing as complex as distributed computing.”

A Note on Distributed Computing
Present
Microservices
Microservices
The Re-emergence of RPC Frameworks
Finagle

RPC System for the JVM

IDL: Thrift

Based on Futures

Request/Response
gRPC is a Multi-Language RPC framework that supports bidirectional streaming. It uses IDL: Protobuufs and supports languages such as C/C++, C#, Node.js, PHP, Ruby, Python, Go, and Java.
Modern RPC Frameworks

Don’t Provide a Unified Model
“The hard problems in distributed computing are not the problems of getting things on and off the wire.”

— A Note on Distributed Computing
The point of RPC was to make remote calls just as simple as local calls.
The point of RPC was to make remote calls just as simple as local calls.

If we treat everything as remote, have we simplified distributed computation at all?
The point of RPC was to make remote calls just as simple as local calls.

If we treat everything as remote, have we simplified distributed computation at all?

If we can't treat all calls as local, is the *procedure call* the right abstraction for distributed computation?
Future

Lasp: Distributed Deterministic Data Flow Programming for Erlang

Consistency Analysis in Bloom: a CALM and Collected Approach

Spores: A Type-Based Foundation for Closures in the Age of Concurrency and Distribution
Spores are small units of possibly mobile functional behavior

Serializable closures with capture controlled by the type system (Scala)

Dual to Actor Systems (like Erlang). Actors exchange data with async messaging, spores are stateless processes that pass functions around with asynchronous messages
Thanks

Resources

https://github.com/CaitieM20/Talks/tree/master/DistributedProgrammingRPC