Implementing Remote Procedure Calls

ANDREW D. BIRRELL and BRUCE JAY NELSON
Xerox Palo Alto Research Center

Remote procedure calls (RPC) appear to be a useful paradigm for providing communication across a network between programs written in a high-level language. This paper describes a package providing a remote procedure call facility, the options that face the designer of such a package, and the decisions we made. We describe the overall structure of our RPC mechanism, our facilities for binding RPC clients, the transport level communication protocol, and some performance measurements. We include descriptions of some optimizations used to achieve high performance and to minimize the load on server machines that have many clients.

CR Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols—*protocol architecture*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications, network operating systems*; D.4.4 [**Operating Systems**]: Communications Management—*message sending, network communication;* D.4.7[**Operating Systems**]: Organization and Design—*distributed systems*

General Terms: Design, Experimentation, Performance, Security

Additional Keywords and Phrases: Remote procedure calls, transport layer protocols, distributed naming and binding, inter-process communication, performance of communication protocols.

1. INTRODUCTION

1.1 Background

The idea of *remote procedure calls* (hereinafter called RPC) is quite simple. It is based on the observation that procedure calls are a well-known and well-understood mechanism for transfer of control and data within a program running on a single computer. Therefore, it is proposed that this same mechanism be extended to provide for transfer of control and data across a communication network. When a remote procedure is invoked, the calling environment is suspended, the parameters are passed across the network to the environment where the procedure is to execute (which we will refer to as the *callee*), and the desired procedure is executed there. When the procedure finishes and produces its results, the results are passed backed to the calling environment, where execution resumes as if returning from a simple single-machine call. While the calling environment is suspended, other processes on that machine may (possibly)

Authors' address: Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304.

on single-user workstations, although it is also used for the construction of servers (shared computers providing common services, accessible through the communication network).

Most of the computers used for Cedar are *Dorados* [8]. The Dorado is a very powerful machine (e.g., a simple Algol-style call and return takes less than 10 microseconds). It is equipped with a 24-bit virtual address space (of 16-bit words) and an 80-megabyte disk. Think of a Dorado as having the power of an IBM 370/168 processor, dedicated to a single user.

Communication between these computers is typically by means of a 3-megabit-per-second Ethernet [11]. (Some computers are on a 10-megabit-per-second Ethernet [7].) Most of the computers running Cedar are on the same Ethernet, but some are on different Ethernets elsewhere in our research internetwork. The internetwork consists of a large number of 3-megabyte and 10-megabyte Ethernets (presently about 160) connected by leased telephone and satellite links (at data rates of between 4800 and 56000 bps). We envisage that our RPC communication will follow the pattern we have experienced with other protocols: most communication is on the local Ethernet (so the much lower data rates of the internet links are not an inconvenience to our users), and the Ethernets are not overloaded (we very rarely see offered loads above 40 percent of the capacity of an Ethernet, and 10 percent is typical).

The PUP family of protocols [3] provides uniform access to any computer on this internetwork. Previous PUP protocols include simple unreliable (but high-probability) datagram service, and reliable flow-controlled byte streams. Between two computers on the same Ethernet, the lower level raw Ethernet packet format is available.

Essentially all programming is in high-level languages. The dominant language is *Mesa* [12] (as modified for the purposes of Cedar), although *Smalltalk* and *InterLisp* are also used. There is no assembly language for Dorados.

1.3 Aims

The primary purpose of our RPC project was to make distributed computation easy. Previously, it was observed within our research community that the construction of communicating programs was a difficult task, undertaken only by members of a select group of communication experts. Even researchers with substantial systems experience found it difficult to acquire the specialized expertise required to build distributed systems with existing tools. This seemed undesirable. We have available to us a very large, very powerful communication network, numerous powerful computers, and an environment that makes building 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 distributed applications. RPC will, we hope, remove unnecessary difficulties, leaving only the fundamental difficulties of building distributed systems: timing, independent 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