Implementing Remote Procedure Calls

Andrew D. Birrell and Bruce Jay Nelson
Xerox Palo Alto Research Center
ACM Transactions on Computer System

Yogi Salomo M Pratama (2015712653)
Supervisor: Professor Seo Euiseong
ECE5658-41 Operating System Design, Sungkyunkwan University
Outline

- Introduction
  - What is RPC
  - Purpose
  - How it Works
  - Code Examples

- Implementation
  - Structure
  - Binding Mechanism
  - Packet-level Protocol
  - Marshalling

- Semantics of RPC
- Exception Handling
- Security
  - Optimization and Performance
    - Optimization
    - Performance Evaluation
  - Conclusion
Introduction
What is RPC?

- A paradigm for accessing network service where client invokes services by making local procedure call

**WHY IS IT ATTRACTIVE?**

+ Clean and simple semantics
  - Easier to build distributed computations and to make it right

+ Efficiency
  - Simple enough for the communication to be quite rapid

+ Generality
  - In single machine computations, procedure are one of important communication mechanism between parts of the algorithm
Purpose

Primary Motivation

• Making distributed computation easy

Secondary Motivation:

• Make RPC communication highly efficient
• Provide secure communication with RPC
How it Works?

Fig. 1. The components of the system, and their interactions for a simple call.
Sample Code (User Application)

A program that displays the number of users on a remote host

```c
#include <rpc/rpc.h>
#include <rpcsvc/rusers.h>
#include <stdio.h>

/*
 * A program that calls the
 * rusers() service
 */

int main(int argc, char **argv)
{
    int num;
    if (argc != 2) {
        fprintf(stderr, "usage: %s hostname\n", argv[0]);
        exit(1);
    }
    if ((num = rusers(argv[1])) < 0) {
        fprintf(stderr, "error: rusers\n");
        exit(1);
    }
    fprintf(stderr, "%d users on %s\n", num, argv[1]);
    exit(0);
}
```
Sample Code
(Client Side)

#include <stdio.h>
#include <utmp.h>
#include <rpc/rpc.h>
#include <rpcsvc/rusers.h>

/* a program that calls the RUSERSPROG
* RPC program
*/

main(int argc, char **argv)
{
    unsigned long nusers;
    enum clnt_stat cs;
    if (argc != 2) {
        fprintf(stderr, "usage: rusers hostname\n");
        exit(1);
    }
    if( cs = rpc_call(argv[1], RUSERSPROG,
                    USERSVERS, RUSERSPROC_NUM, xdr_void,
                    (char *)0, xdr_u_long, (char *)&nusers,
                    "visible") != RPC_SUCCESS ) {
        clnt_perrno(cs);
        exit(1);
    }
    fprintf(stderr, "%d users on %s\n", nusers,
            argv[1]);
    exit(0);
}
```c
#include <stdio.h>
#include <rpc/rpc.h>
#include <rpcsvc/rusers.h>

void *rusers();

main()
{
    if (rpc_reg(RUSERSPROG, RUSERSVERS,
                 RUSERSPROC_NUM, rusers,
                 xdr_void, xdr_u_long,
                 "visible") == -1) {
        fprintf(stderr, "Couldn't Register\n");
        exit(1);
    }
    svc_run(); /* Never returns */
    fprintf(stderr, "Error: svc_run returned!\n");
    exit(1);
}
```

Sample Code (Server Side)

Register procedure to be called and invoke those procedures
RPC Implementation
Structure

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

Using interface module: list of procedure names, with types of arguments and results

Programmer does not need to build detailed communication-related code
Binding Process: Naming

How does a client of the binding mechanism specify what he wants to be bound to?

- Remote procedures are referred with **interface**

Two parts of interface name:

**Type**: Specify which interface the caller expects the callee to implement
- Example: Correspond to abstraction of “mail-server”

**Instance**: Specify which particular implementor of an abstract interface is desired
- Example: Correspond to some particular mail server
Binding Process: Locating

How does a caller determine the machine address of the callee and specify to the callee the procedure to be invoked?

- Using **Grapevine Distributed Database**
  - Widely and reliably available
  - At least three copies of each database entry

- Alternative ideas:
  - Include callee network address
    - Too early binding for most applications
  - Broadcast Protocol
    - Too much interference with innocent bystanders
  - Not convenient for local network

<table>
<thead>
<tr>
<th>Type (Group)</th>
<th>Member-list</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileAccess</td>
<td>{Ebbets, Luther}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance (Individual)</th>
<th>Connect-site</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ebbets</td>
<td>3#22#</td>
</tr>
<tr>
<td>Luther</td>
<td>3#276#</td>
</tr>
</tbody>
</table>
Binding Process: Locating

Exporting & Importing an Interface
What should client specify when it want to import an interface?

- **Only the type** of the interface and not its instance
  - The decision about the interface instance is made dynamically

- **The RName**
  - Delaying the choice of a particular exporting machine

- **Network address**
  - Most restrictive
  - Done at compile time

- **Dynamically instantiate interface** and import them
  - Allowing importer to bind his program to several exporting machines even when it cannot know statically how many machines it wishes to bind to
Packet-level Transport Protocol

What protocol should be used?

- RPC specific protocol is implemented
  - Stream protocol, such as TCP, focused on **high throughput**. In RPC, **low latency** is more important than high throughput.
  - Performance gains of a **factor of ten**
- The requirements (aim) in the protocol design:
  - Minimizing the elapsed real-time between initiating call and getting results
  - Minimize load imposed on a server by substantial numbers of users
  - Guarantee that if the call returns, the related procedure in the server has been invoked precisely once
Fig. 4. A complicated call. The arguments occupy two packets. The call duration is long enough to require retransmission of the last argument packet requesting an acknowledgment, and the result packet is retransmitted requesting an acknowledgment because no subsequent call arrived.
Marshalling

How does the client delivered and represented the parameters of the procedure?

- The types of parameters must be known by modules that do conversion
- Some types might need to be interpreted/converted
  - 4-byte integer type may need to be bit-reversed

Parameters

Simple Type
- Integer

Compound Type
- C struct
Semantics of RPC

How will RPC handle failures?

- Considerations: Operations that can safely be executed twice (*idempotent*)
  - Example: Fetching current time, Retrieving particular page of a file

<table>
<thead>
<tr>
<th>Exactly once</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Most desired, but cannot achieved at low cost</td>
</tr>
<tr>
<td>- If the client transmits a request, and the server crashes, the client has no way of knowing whether the server had received and processed the request before crashing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>At most once</th>
</tr>
</thead>
<tbody>
<tr>
<td>- When control returns to the caller, the operation will have been executed no more than once</td>
</tr>
<tr>
<td>- If the server crashes, the client will be notified of the error, but will have no way of knowing whether or not the operation was performed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>At least once</th>
</tr>
</thead>
<tbody>
<tr>
<td>- The client just keeps retransmitting the request until it gets the desired response</td>
</tr>
<tr>
<td>- On return to the caller, the operation will have be performed at least one time, but possibly multiple times</td>
</tr>
</tbody>
</table>
Exception Handling

How would RPC handle Exception?

- Two level of exception:
  - Call failed exception
    - Raised by RPCRuntime when there is some communication difficulty
    - Primary difference between local and remote calls
  - RPC Exception
    - Handled by RPCRuntime in the caller machine by raising exception in the appropriate process
    - If there is catch phrase, it is executed
    - If the catch phrase returns, the results are passed back to callee machine and event proceed normal
    - If the catch phrase terminates by a jump, the callee machine is also notified
Security

What security measures will be provided by RPC?

- Provide encryption-based security for calls
  - Protection from eavesdropping and concealing patterns of data
  - Detecting attempt at modification, replay, or creation of calls
- Using Grapevine as authentication service
- Restricting set of users who can update Grapevine
Optimization

and

Performance
Optimization Methods

▪ Usage of subsequent packet for implicit acknowledgment of previous packet

▪ Minimize cost of establishing and terminating connections

▪ Use of idle processes in caller and callee machines to reduce process creation and swaps

▪ Bypassing software layers by implementing packet-level protocol
Performance Evaluation

▪ Scenario:
  ▪ Remote calls between two Dorado machines connected by Ethernet
  ▪ 12,000 calls on each procedure
  ▪ Network is shared with other users but lightly loaded (5-10% capacity)
  ▪ Measuring elapsed time: Including time spent waiting for the network and time used by interference from other devices
  ▪ Elapsed time calculated by: Dorado microprocessor cycles / known crystal frequency (accuracy 10%)
For transferring large amounts of data in one direction, other methods have advantage over RPC

No performance measurement of exporting and importing interface yet
Conclusion

- RPC is one of the techniques used in communication between processes outside their space address boundaries
- RPC Protocol avoids communication overhead
- One of the first protocol that provide security in network communication
- Succeed in providing reliability and integrity for data and control transfer
Thank you