Remote Procedure Call: Limitations and Drawbacks

Raghav Kukreja & Nitin Garg
Department of Information Technology, Dronacharya College of Engineering, Gurgaon, India
Email: nitingarg2015@gmail.com

ABSTRACT

The remote procedure calls are broadly used in distributed operating systems. Since it is easy and simple to implement, its usage is wide in the distributed operating systems. It also has some drawbacks and limitations. Though very subtle, these drawbacks are of utmost importance. In this paper we discuss problems with Remote Procedure Call (RPC) in the area of the concept of the model itself, technical problems with its implementation, and problems of client and server crashes and some of the performance issues. The paper discusses these problems and the proposed solutions of the problems.

INTRODUCTION

RPC is assumed to be apt paradigm for creating the distributed operating system. It is a way of communication between the two parties, the client and the server. For example, assuming that computation consists of main program, running on client machine, and the procedure being called, runs on server machine. When main program calls a procedure, what happens is that the call is made to a procedure called client stub on client’s machine. The client stub collects parameters into the message, and then sends that message to Server machine where it is being received by server stub. The server stub unpacks parameters from that message, and then calls the server procedure using standard calling sequence. In this way, both main program and called procedure see ordinary, and local procedure calls, using regular calling conventions. Only those stubs, which are automatically generated by compiler, know that call is remote. We discuss various problem shere that researchers have observed with RPC.

CONCEPTUAL PROBLEMS IN RPC

Conceptual problems of the RPC are given as follows:

Which is Server and which is Client?
RPC is inappropriate for all computations. The example where it is inappropriate, assume this UNIX pipeline:
Sort<infile1 | unit | wk -l > outfile1
Thatsort’sinfile1, ASCII file having one word in a line, removes the duplicates, and prints word count in outfile1.It’s hard to know which is client and which is server here. A possible configuration is to have each of three programs to be client and server, divide in two processes if needed. The client parts of sort would send read requests to the file server in order to acquire blocks of input file. Client part of unit would send
requests to server part of the sort in order to provide the sorted data as it becomes available. Client part of that would send request to server part of the inverse in order to provide the duplicate less data as it becomes available.

**Unexpected Messages**
Many situations may exist where one of the process may have an important information for the other process, but a client, is not expecting such information. It remains hard for holder of information to transmit it. In virtual circuit model having full duplex connections, any party may send the high-priority message at any given moment. Though, problem arises when the user at the terminal issues command and then afterwards realizes that command has to be aborted, for example, if the given program starts up is debugged and has got in the infinite loop. If user presses there or the BREAK key, the character might be held by terminal server till someone asks for it, which never really happens. What needed here is a way for server, which is usually passive, to initiate the action in order to signal runaway command or shell that was used to start it. Here is an example of a process which is usually a server but in special cases needs to act like client and communicate with the other processes which might not be expecting any communication. While there are various ad hoc solutions which may be devised, so RPC model is faulty here.

**Multicast**
Situations usually exist where one process may want to send message to other processes. Local area networks are usually able to support the broadcast or the multicast in the hardware. Packet transmitted in the broadcast or the multicast mode might be received by the multiple machines. Hence, there is a situation where the processes may need to do the multicasting and hardware doing it. An RPC approach is a two-party communication, hence, there is no way to use the facilities of the hardware.

**TECHNICAL PROBLEMS**
Technical problems regarding the RPC are as follows:

**Parameter Marshalling**
In order to arrange the parameters, client stub needs to know the number and type of them. For strong languages, they do not cause trouble, though in case the union type is permitted, stub does not deduce which union member is passed. For languages like C, which is not type safe, problems are worse. The keyword print can be called with number of different parameters. In case print is called remotely, client stub does not have easy way of offending out about the number of parameters there are or the types of these parameters.

**Parameter Passing**
When client calls the stub, call is made by normal calling procedure. Stub collects parameters and inserts them into message that is being sent to server. In case, all parameters are the value parameters, problems do not arise. They get copied in the message. But, if reference parameters or pointers are there, things get complicated. It is easy to copy pointers in message, while the server wants to use them, it would not work right because object pointed would not be available.
Global Variables
The programming languages may offer programmer ways to declare a global variable. Some of the procedures might access global variables simply by using them. In case, procedure which originally was build to run locally is forced to run remote may have references to global variable, this would fail and procedure would not work. Such problem is same as that of the pointer variables and is as hard to deal with.

ABNORMAL SITUATIONS
Till now we have assumed that client or server never crashes. That would be very optimistic of us to believe so. Lots of distributed systems are build to be tolerant to faults, so they try to reboot crashed processors by themselves.

EXCEPTION HANDLING
Usually when a main program wants to calls procedure, in case, code is correct the procedure returns to caller. In case, the machine crashes, main program and procedure die and whole program needs to run again. Hence, there are two modes of operation: whole program works or whole program fails.

Repeated Execution Semantics
When local procedures recalled, they are executed only one time. RPC having one execution semantics is not possible to achieve in real case, it is also complicated and expensive in various cases.

Loss of State
When a server crashes between RPCs and reboots before next RPC, severe problems may occur. Such problems are because of fact that server has a long term state data or information of client. File server might have the information of the open files, position of current files etc, will be lost after reboot of the server. When client tries to read the file 0, server would not have any idea of what file is to be read or how far the was the program.

Orphans
In case, the client crashes and server is not idle, computations of the server become orphan. Virtual circuit systems do not have to suffer with this problem because when the client crashed, the circuits get broken and this is found out by the server, which kills all the computations initiated by client.

PERFORMANCE PROBLEMS
Performance problems regarding the RPC are given as follows:

Lack of Parallelism
In RPC either the server is active or the client is active at a time. Hence, there is no parallelism which is possible. Clients and the servers are co-routines. In other models it might be possible to run the computations simultaneously between server and client but not in RPC.

Lack of Streaming
In database a client requests a server to perform the operation in order to look up data in the database that meet some conditions. Using this model, a server should wait till all information are found before replying. In case operation of searching all data is time consuming, client
remains idle for some time, waiting for last tuple to be searched.

**Bad Assumptions**
In some cases, programmers may utilize procedures rather than inline code because it is much more modular and will not affect performance. For example, some programs have negligible routine to exchange element with element $I$. In case, such a procedure is run remotely, it will slow down whole computation by the ratio of probably a factor of thousand. With no transparent communication, procedure would never run remote.

**Conclusion**
The remote procedure calls though easy to implement, have many limitations and drawbacks. Even though the drawbacks and limitations are subtle and negligible in comparison to its advantages and importance, it cannot be neglected. Different issues such as technical problems and performance issues were discussed in this paper. The paper has discussed these problem areas in brief and these problems shall be looked upon by the researchers in the future.

**References**


