European Journal of Scientific Research
ISSN 1450-216X Vol.27 No.1 (2009), pp.112-119
© EuroJournals Publishing, Inc. 2009
http://www.eurojournals.com/ejsr.htm

Interactive RPC Binding Model

Fawaz A. M. Masoud
Department of Computer Science, King Abdulla II School for Information Technology
University of Jordan, Amman, Jordan
E-mail: fawaz @ju.edu.jo

Qatawneh Mohammad
Department of Computer Science, King Abdulla II School for Information Technology
University of Jordan, Amman, Jordan
E-mail: mohd.qat@ju.edu.jo

Wesam Almobaideen
Department of Computer Science, King Abdulla II School for Information Technology
University of Jordan, Amman, Jordan
E-mail: wesmoba@ju.edu.jo

Azzam Sleit
Department of Computer Science, King Abdulla II School for Information Technology
University of Jordan, Amman, Jordan

Amjad Hudaib
Department of Computer Science, King Abdulla II School for Information Technology
University of Jordan, Amman, Jordan
E-mail: ahudaib@ju.edu.jo

Alshraideh Mohammad
Department of Computer Science, King Abdulla II School for Information Technology
University of Jordan, Amman, Jordan
E-mail: mshrideh@ju.edu.jo

Nabil Abu Hashish
Department of Computer Science, King Abdulla II School for Information Technology
University of Jordan, Amman, Jordan
E-mail: nabilm@ju.edu.jo

Oraib Megdady
Department of Computer Science, King Abdulla II School for Information Technology
University of Jordan, Amman, Jordan
E-mail: orayb_miqdadi@yahoo.com

Shatha Al-Asir
Department of Computer Science, King Abdulla II School for Information Technology
University of Jordan, Amman, Jordan
E-mail: sh.alasir@gmail.com

Shorouq AlAdaileh
Department of Computer Science, King Abdulla II School for Information Technology
University of Jordan, Amman, Jordan
E-mail: shorouq_adaileh@yahoo.com

Abstract

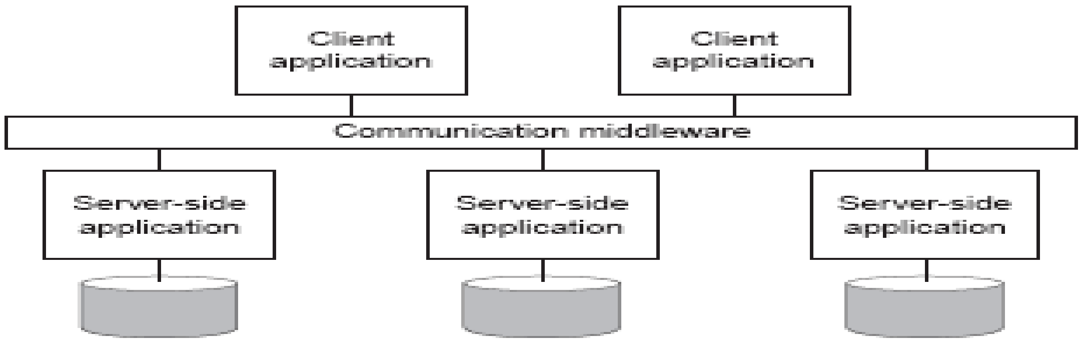
This paper proposes a new Remote Procedure Call (RPC) binding model, in which some interaction between the three parties of the system occurs (client, server and binder). The new model uses the server status and the load volume parameters to be exchanged between the parties. The experimental results revealed that the new model is able to decrease the number of retransmitted RPCs due to response time out increasingly when the request rate increase, and decreased the average execution time for a RPC (time needed to send RPC and receive the reply). Moreover, it was also able to increase the load balancing between the servers.

Keywords: Remote Procedure Call, Binder, Client, Server.

1. Introduction

Distributed systems are constructed of multiple independent computers that appear to its user as a single coherent system. The client-server model is a standard model for these systems, where the client might request a procedure that exists on another server in the network. Using a middleware communication channel clients would be able to send their Request to the servers and receive a reply as shown in figure 1, such communication process is called Remote Procedure Call (RPC) [8].

Figure 1: RPC network [8]

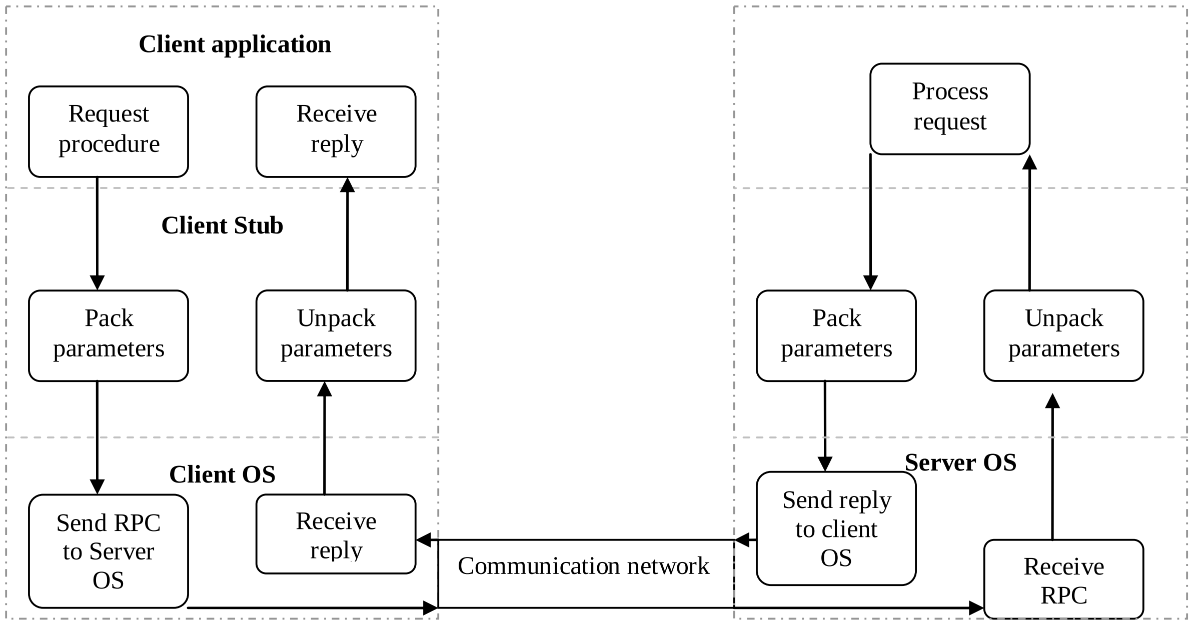


Remote procedure call is a client-server mechanism that enables an application on one machine to make a procedure call to execute on another machine in a network without having to understand network details. Such mechanism is achieved though a common interface, which is called middleware.

An RPC is similar to the local procedure call (LPC) model. In the local procedure call, the caller places parameters to a procedure in some well-defined location such as a register window. It then transfers control to the procedure, and eventually regains control. At that point, the results of the procedure are extracted from the well-defined specified location, and the caller continues execution. While in the RPC model as shown in figure 2, the client calls a local procedure- a stub routine- that packs its parameters into a message and sends them across the network to a particular server process and waits (blocks) for a reply message. The call message includes the procedure's parameters, and the reply message includes the procedure's result. Once the reply message is received, the results of the procedure are extracted, and the caller's execution is resumed.

Meanwhile, the server process unpacks the message, calls the procedure, packs the return results into a message, and sends them back to the client stub. The client stub unblocks receives message, unpacks the results of the RPC, and returns them to the caller. This packing of parameters is called marshaling.[7][8]

Figure 2: Client - Server RPC procedure



1.1 Binding

Binding is the process of connecting a certain client process to a certain server process. It is not considered as a part of RPC. Binding information is used to inform the client how to find and connect with the desired server. Distributed systems often use one of the following binding methods:

1. Automatic binding, in which binding is performed automatically by the client stub. The client stub passes the binding handle after getting it from name service database to the RPC run time library.
2. Implicit binding, where an application code should be written to obtain binding information and binding handle is performed in a global area inside the client stub.
3. Explicit Binding, where the binding handle should be passed explicitly when the RPC is called, the application code should be written explicitly too, in order to set the binding handle.

However, for a client to bind to a server there are multiple mechanisms, the simplest one is to use a hard coded server name, but this method is inflexible, because if any change occurred on the server name, version or the location, the client code need to be upgraded, this problem was solved by using the directory service,[7][3][8]

The other mechanism is to use a directory service, when servers start; they advertise about them self and their services (interfaces) in the directory service. Then, if a client requires a service it refers to the directory service to get the location of the required interface. Directory service model offers more flexibility, as if a server move or there were more than one server implements the same service, directory service will tolerate balancing request.

2. Related Work

RPC binding and directory service has been discussed in many distributed systems models and papers. Due to the gap between the speed on LPC and RPC, many suggested solutions were proposed, [2][8][4]. In this section, we present some of those papers.

The Authors of paper [2], considered the enormous performance gap between the LPC and RPC, and designed a flexible linker (FLink) to achieve a sufficient switching model between the various binding techniques, they also suggested that the choosing the binding technique should be dynamic depending on the situation of the OS. FLink provides three different binding techniques that allow the operating system to flexibly balance with their cost. The experimental results of the paper showed that the new flexible linker wasn't just able to improve distributed system performance, but it also was considered as an enabling technology to new binding strategies, because it provided a close integration of compile-time and run time.

Also the authors of paper [8], presented a new way of searching remote procedures, based on the GridRPC systems and OGSA. The technique suggested using a grid enabled RPC model and procedure search mechanism in OGSA to help discovering the RPCs in grid computing implementation instead of using the registry service. In the new technique based on the gridRPC the clients can obtain some procedures that agree with their requirement without the need to refer to the registry service, which will certainly reduce the overhead of querying the registry service upon each request.

The Authors of paper [4], proposed a RPC new mechanism that will increase the RPC fault tolerance and increase the parallelism between the network communication model and the executed RPC on the servers side, the suggested mechanism has also enhanced communication error detection, and reduced the occurrence of orphan messages

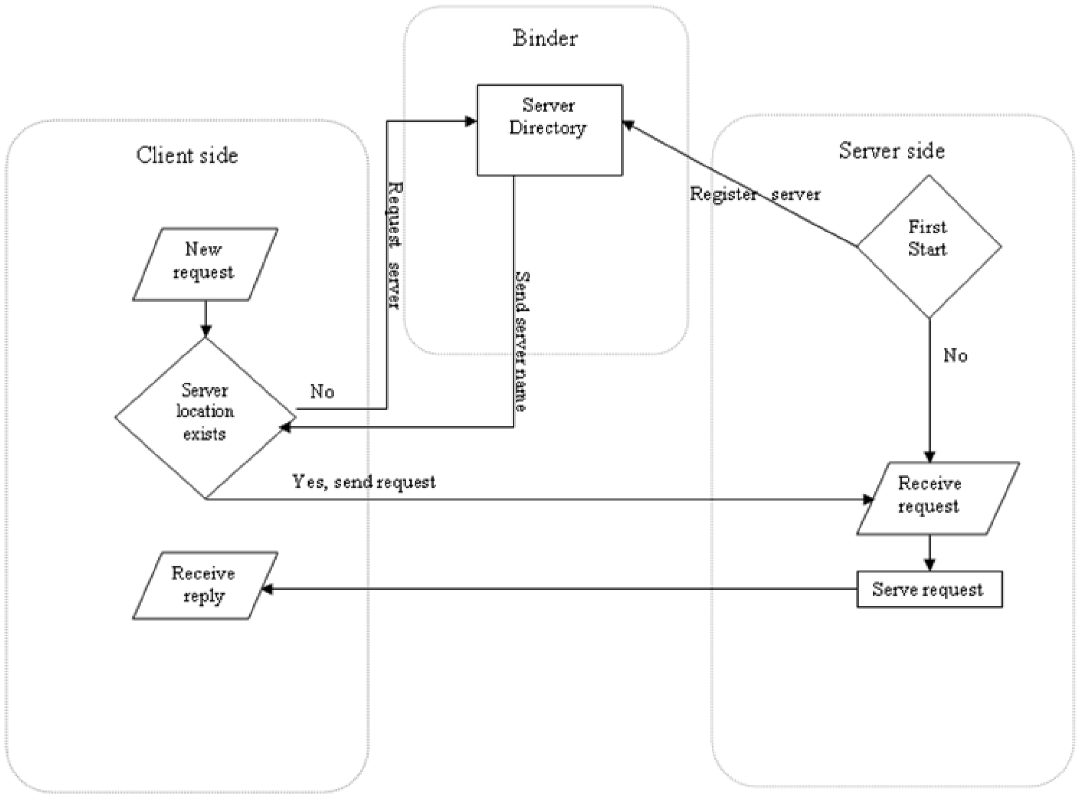
As seen in the previous works, all the papers suggest new model and mechanisms in order to enhance the RPC model, and mind the gap between the RPC and the LPC. In our research also we try by proposing a new binding model to reduce the gap by reducing the RPC execution time (request _ reply).

3. Proposed Model

Figure 3 shows the RPC binding model, the binding procedure between client and server could be summarized in the following steps:

1. When server starts, it advertises about its services to the binder (register to the binder).
2. When a client requests a service for the first time, it sends a server request to the binder, which sends back server information to the client.
3. Client saves the server information for later use, and sends the request to server.
4. Later each time the client request the same service, it uses the information stored in its cache.

Figure 3: RPC binding model



In this model, the client keeps using the stored information, until the server crashes (it goes down, or it becomes overloaded, and unable to accept more requests), [7] [8].

Even though, directory server(binder) performs some load balancing when sending servers information to clients, the balancing operation has a limited ability to keep the balanced distribution for the RPC, because the load balancing operation is done only if client doesn't have any server information for a specific service, and send a request to the binder, but with time the need to refer to directory server(binder) decrease, and the communication process will be done directly between the server and the client.

However, in highly interactive distributed systems, the rate of sending request are very high, and many clients may send their requests to the same server, with the time the server becomes overloaded and RPC response time will incredibly increase.

In this research we propose some modification on the binding procedure, we suggest sending more control messages between the clients, servers & directory server (binder), in order to achieve more load balancing and decrease the retransmission due to time out procedure.

Two types of control message are suggested to be sent, periodically or triggered.

- The first type of control messages is sent, as a parameter added to the server registration message, this parameter tells the service directory (binder), how much the server is loaded, this parameter is calculated according to the number of waiting RPC in the server buffer, and the estimated time to finish these requests. Although the server status parameter will increase the size of the message (one byte), it will surely help the binder to do more balancing when distributing the services locations (server addresses), this message is sent periodically, or triggered when the server become overloaded.
- The second type of control message is sent from the server to the client. When the server sends back a reply for a RPC it sends also a parameter telling the client about its status (how

much it is loaded), if the server was highly loaded, the client will remove it from its records, and the next time it need send the same RPC it will send a server request to the binder, in this way the binder will be able to give it a suitable server, according the information it has about all the other servers.

4. Results and Analysis

The proposed model was tested and compared to the standard model, using a computer lab that consists of: one router and one switch connected to computers with the following specifications: (Pentium 4, CPU 3.00 GHz, 512 RAM), and the maximum distance between PCs was 15 meters.

The comparison between the two models was done according to the: number of retransmitted RPCs due to response time out, and the average execution time (time needed to send RPC and receive a reply), we executed the model using the input configuration shown in table 1:

Table 1: Experiment input configuration

Configuration:	**Value**
Types of RPC	10
Number of servers	3
Number of services on each server	7
Number of RPCs sent by each client	50
Time out value	5000 ms

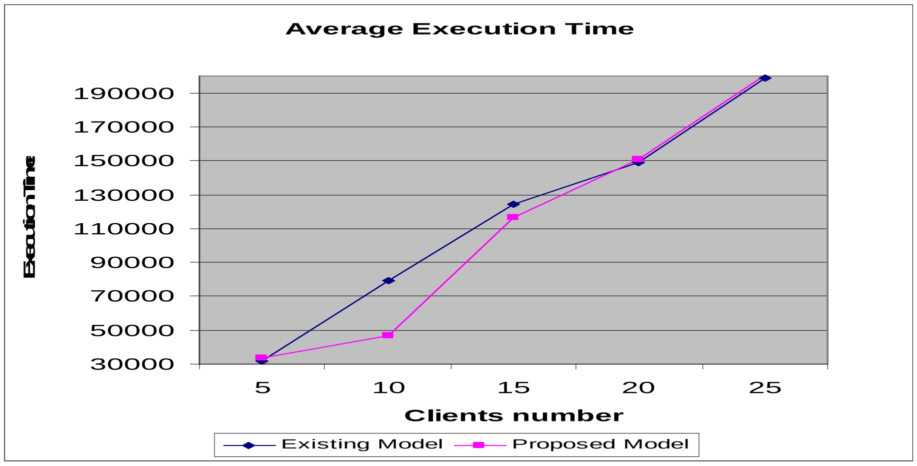
The general scenario used in the experiments was as follows: each client in the system use synchronous transmission mode, and has 50 request to be sent, the type of request is chosen randomly, and the request rate average equals (500ms).

The experiments were applied using on five different scenarios, each of which was executed 6 times:

The first scenario used 5 clients, and the total number of sent RPCs was 250, the other scenarios used 10, 15, 20 and 25 clients while the numbers of sent RPCs in the scenarios were 500,750, 1000 and 1250 Request relatively

As shown in the figure 4 the new model was able to decrease the average execution time for the RPC monotonically, when the number of clients increased, that is due to the less number of packets retransmitted when the servers was overloaded.

Figure 4: Average execution time & number



When the number of clients was 5, the new model led to higher execution time compared with the standard RPC model; that could be referred to the over head, caused by the server load calculations, and the periodically sent registration messages, which caused an over head, because the system is stable and the load on the servers is relatively low.

When the number of client increased (10-15) clients the number model was able to decrease the execution time, because calculation process and the control messages wasted time were low compared to the load on the servers.

However, the balancing between the number of servers in the system and the number of clients is very important, in order to gain better performance because even when applying the new model on a system where the number of servers is low compared to the number of client and the requests generation rate, the execution time will not be decreased because all the servers will be overloaded, even when applying highly load balancing techniques, we were able to notice that from the results of using 20 and 25 clients as shown in figure 4.

Figure 5: Number of Timeout request and number of clients

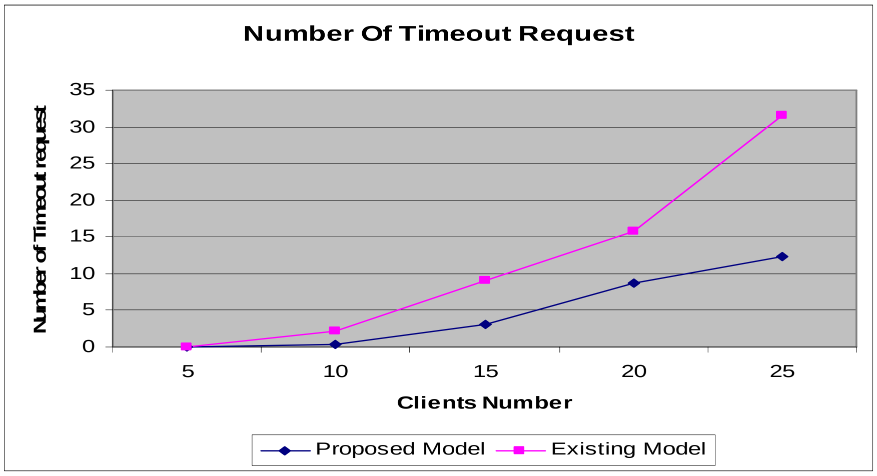


Figure 5. shows the number of retransmitted RPCs using the standard RPC model and the modified one, as the figure illustrates, our model has decreased the number of retransmitted RPCs due to client time out, and that’s because more balancing where performed in distributing the RPCs over the servers, which increased the ability of the server to process the request and send a reply faster than the original system.

5. Summary and Conclusions

In this paper, we proposed a new binding model, in which we suggested sending more control parameters between clients and servers and between Directory server (binder) and servers. Messages from server to binder are sent triggered or periodically, while the messages from the server to the client are sent within each reply. We have tested our new model and compared it to the standard RPC model, using a distributed system lab, the results of implementing the new model led to:

- Higher load balancing in distributing RPCs between the servers.
- Decrease the number of retransmitted RPCs because the probability of sending a RPC for a highly loaded server was decreased.
- The model was able to decrease the average execution time for the RPCs, because the number of retransmitted RPCs decreased and load over the servers where more balanced. However the new model was able to decrease the average execution time up to a certain limit only, because

number of servers to the number of clients was really small, which certainly led to increase the overload on all the servers.

References

[1] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, H. Casanova, "*GridRPC: A Remote Procedure Call API for Grid Computing*", GWD-I (Informational) Advanced Programming Models Research Group, http://www.eece.unm.edu/˜apm/docs/APM GridRPC 0702.pdf, July 2002.

[2] Markus Pizka, "*Improving Distributed OS Performance by Flexible Incremental Linking*", Boltzmannstr, citeseer.ist.psu.edu/630883.html, 2002.

[3] R. Srinivasan, "*RPC: Remote Procedure Call Protocol Specification*" Version 2, Network Working Group, , Sun Microsystems, 1995.

[4] Shiva S.; Virmani R; "*Implementation of reliable and efficient remote procedure calls*", Southeastcon apos;93, Proceedings, IEEE Volume, Issue 4-7 Page(s):5 Digital Object Identifier 10.1109/SECON.1993.465780, 1993

[5] Tang Chen, Dwarkadas Scott, "*Integrating Remote Invocation and Distributed Shared State*", citeseer.ist.psu.edu/tang04integrating.html, 2004.

[6] Wanlei ZhouA1 and Andrzej GoscinskiA, "*Managing Replicated Remote Procedure Call Transactions*", The Computer Journal 1999 42(7):592-608; doi:10.1093/comjnl/42.7.592 © 1999 by, Volume 42, Number 7 Pp. 592-608, 1999.

[7] "*The RPC chapters of the* IBM DCE for AIX", Version 2.2: Application Development Guide and the IBM DCE for AIX, Version 2.2: Administration Guide, http://www.univie.ac.at/dcedoc/A3U2H/A3U2HM02.HTM#ToC.

[8] Yue-zhuo Zhang , Yong-zhong Huang and Xin Chen, "*A Procedure Search Mechanism in OGSA-Based GridRPC Systems*" Springer Berlin / Heidelberg, pages 400-403, Session 7: Network Communication and Information Retrieval, 2004.