Yugoslav Journal of Operations Research
6 (1996), Number 1, 121-127

THE ACTIVE SIDE PRINCIPLE APPROACH TO THE DESIGN OF CLIENT SERVER PROTOCOL

Miroslav HAJDUKOVIĆ, Danilo OBRADOVIĆ, Branko PERIŠIĆ
Faculty of Technical Sciences,
University of Novi Sad,
Fruškogorska 11, 21000 Novi Sad,
Yugoslavia

Abstract: This paper discusses possible improvements to the protocol intended to support the client server model of process cooperation inside a distributed computer system. Protocol improvements are possible by applying the active side principle, based on the optimistic approach. This leads to the design of an active side protocol with better characteristics with respect to the exchange of smaller numbers of messages and more efficient implementation, compared to protocols of the same class.

Keywords: Distributed computer system, client server model, remote procedure call, protocol.

1. INTRODUCTION

The client server model is the typical form of process cooperation inside a distributed computer system [4], [5], [7]. This model imitates the program subprogram relationship, with the difference that a program appears as a process on one computer, and a subprogram as a process on another computer of the same distributed computer system. It is not strange that a variant of the client server model is known under the name remote procedure call [1], [6].

The aim of the client server model is to make it possible for a process client, active on one computer to get service from a process server, active on another computer of the same distributed computer system. This kind of cooperation is supported by three primitives: request, expect and answer. The first is intended for the client to inform the server about the requested service and to wait for an answer from the server. The second is used by the server to wait for the request, and the third enables the server to send the answer to the client.

The client server model of process cooperation is based on message exchange under conditions of unreliable communications, with the consequence that all messa-

ges do not reach their destinations. The causes of unreliability lay in data transfer errors, malfunctions which stop the activity of destination processes or discrepancy in speeds of source and destination processes (resulting from the different throughputs of their computers). Implementation of the three basic primitives of the client server model must be backed up by a protocol designed to provide message delivery whenever there are at least minimal conditions for this.

2. DESCRIPTION OF THE CLIENT SERVER PROTOCOL

The client server protocol coordinates the behavior of process client and process server sides, and provides successful message exchange between them under unreliable data transfer conditions.

A request primitive call activates the protocol on the process client side. This sends the request message and blocks the process client until the answer message arrives. If this message is not received within the expected time, multiple retransmissions of the request message are tried before giving up on the answer message. In that case, the request primitive call is unsuccessful.

Expect and answer primitive calls activate the protocol on the process server side. The first call blocks the process server until the request message arrives. After that, the server process serves the received request, and, at the end, it calls up the answer primitive. This leads to sending the answer message, and, eventually, to temporary blocking of the process server (which preserves the answer message for retransmission, if it has not reached its destination).

The client server protocol has to coordinate two sides of behavior under circumstances where each side can conclude about its partner's state only by message arrival or non-arrival. For example, non-arrival of the answer message may be the result of:

1. loss of the request message (this means that the server has not started service and has not sent the answer message)
2. lasting service (because of a server overload)
3. loss of the answer message.

The client cannot distinguish between these three cases, and ought to retransmit the request message, although it is sensible only in the first and the third case. The client server protocol has to cancel retransmited but already received request messages in order to ensure at most one service per request.

3. IMPROVEMENT APPROACH TO THE CLIENT SERVER PROTOCOL

Client server protocol improvement (described in this paper) is directed towards usage minimization of distributed computer system resources during process cooperation, but without sacrificing cooperation efficiency. Experiences with existing

distributed computer systems [6], [8], [3] show that a processor is a much more critical resource for message exchange than a communication channel. It appears that message transmission time (when the communication channel is engaged) is much smaller than message transmission preparation time (when the processor is engaged).

This directs our attention to reducing transmission preparation time, for a great deal of it is spent on message data copying.

Time spent on message transmission preparation can be reduced by cutting down the total number of necessary messages, by sending control instead of data messages (to avoid data copying), and, to some extent, by protocol simplification.

The active side principle, based on the optimistic approach, can help to minimize the total number of necessary messages. The optimistic approach exploits the fact that the overwhelming majority of messages reach their destinations without trouble. This means that the protocol would foresee a minimal number of messages for normal circumstances (when all messages reach their destinations trouble-free). The active side principle states that the active side (or at least the side to become active) should always initiate message sending. This is the case in the normal situation, when the client (as the active side) initiates communication by sending the request message, while the server (as the passive side) is waiting for it. Then, client and server roles are switched, and the server (as the new active side) sends the reply message, while the client (as the new passive side) waits for it. In the case of failure of the transmission channel, the active side principle is usually respected, too. So, the active side tries to retransmit its message. This means that the client retransmits its request message, and the server its reply message. But, when the server is overloaded, the usual approach [4], [5], [1] does not follow the active side principle, as the client (while being passive) periodically asks for message replay. This disturbs the server, because it must answer by acknowledgment to show its activity. A better approach (suggested by the active side principle) is that the server, for longer lasting services, periodically sends "working" control messages, to calm down the client. This causes a 50% reduction in messages compared to the usual approach.

The application of the active side principle based on the optimistic approach makes possible the usage of control instead of data messages, and offers simpler protocol (with a smaller number of protocol participant states during communication), which enables more efficient implementation.

4. PROPOSITION TO IMPROVE THE CLIENT SERVER PROTOCOL

According to the active side principle, the client (as the introductory active side) sends the request message, and, after that, becomes the passive side. After reception of the request message, the (up to then blocked) server becomes the active side. So, in the case of longer lasting service, its duty is to prevent retransmission of the request message by periodic sending of the "working" control message (to calm down the waiting client, and to divert it from any retransmissions). At the service end, the server sends the answer message, and becomes the passive side. Reception of the answer message converts the client to the active side. So, it sends the acknow-

ledgment control message (to eventually shorten the duration of server blockade inside the answer primitive call). In the case of absence of either the "working" control message or the answer message (but after reception of at least one "working" control message), the client reacts by sending the "result?" control message (in an attempt to eventually provoke retransmission of the answer message, if it has been lost). In any case, reception of the first "working" control message is a certain sign that the request message has been received, so its retransmission is superfluous, and would be replaced by the "result ?" control message.

Table 1 and Table 2 contain a complete overview of the proposed client server protocol, named after the active side principle, which inspired its design. The active side protocol is designed strictly along the active side principle. The optimistic approach to active side protocol design causes in normal circumstances only three messages to be exchanged (request, answer, acknowledgment) per short services. For long services a few "working" control messages are needed, too. Request message retransmissions are replaced by sending the "result ?" control messages as soon as possible, and in any case the number of retransmissions is limited. The client is waiting for an answer while it receives "working" control messages.

The active side protocol has practically only one working state (waiting for answer) on the client side (related to a request primitive call), and three working states on the server side (one related to an expect primitive call and two related to an answer primitive call).

5. DISCUSSION OF THE ACTIVE SIDE PROTOCOL

The advantages of the active side protocol can be shown by comparing it to similar protocols, developed for famous distributed computer systems such as V [3], [2], AMOEBA [5] or CEDAR [1].

In normal circumstances, the active side protocol, the V protocol and the CEDAR protocol tend to exchange a similar number of messages per short services, while the AMOEBA protocol foresees acknowledgment of request messages [10].

None of referent protocols completely respects the active side principle. So, for longer lasting services, the blocked client retransmits the request message or an adequate control message, and the busy server reacts by retransmission of a backing control message, which is double, compared to the number of messages sent in the same situation by the active side protocol [10].

With regard to the number of states (protocol simplicity) the active side protocol is similar to the V protocol (it has one client and three server states) and the CEDAR protocol (it has two client and two server states). The AMOEBA protocol is more complex (it has four client and four server states) [9].

6. CONCLUSION

Improvement of the protocol for the client server model of process cooperation inside a distributed computer system is due to the reduction of processor participation in message exchange. This is achieved by application of the active side principle, based on the optimistic approach. The result of this is an active side protocol with better

properties (smaller number messages and more efficient implementation) compared to protocols developed for a famous distributed computer systems.

Table 1. Overview of the active side protocol (the client side)

THE CLIENT SIDE		
STATE	EVENT	ACTION
outside of protocol	any message reception	ignore
	a request primitive call	request message sent, with transition into state "waiting for answer with retransmissions"
waiting for answer with retransmissions	answer message reception	acknowledgment control message sent, and end of waiting inside the successful request primitive call, with transition into state "outside of protocol"
	"working" control message reception	transition into state "waiting for answer without retransmissions"
	any other message reception	ignore
	time for retransmission	request message retransmission or end of waiting inside the unsuccessful request primitive call, with transition into state "outside of protocol"
waiting for answer without retransmissions	answer message reception	acknowledgment control message sent, and end of waiting inside the successful request primitive call, with transition into state "outside of protocol"
	"working" control message reception	postponement of periodic check
	any other message reception	ignore
	periodic check	"result?" control message sent or end of waiting inside the unsuccessful request primitive call, with transition into state "outside of protocol"

Table 2. Overview of the active side protocol (the server side)

THE SERVER SIDE		
STATE	EVENT	ACTION
outside of protocol	any message reception	ignore
	an expect primitive call	transition into state "waiting for request"
waiting for request	request message reception	ignore old (served) messages or end of waiting inside the expect primitive call, with transition into state "servicing", and starting of service
	any other message reception	ignore
servicing	any message reception	ignore
	time to calm down the client	send of "working" control message
	an answer primitive call	send of answer message, with transition into state "waiting for acknowledgment"
waiting for acknowledgment	acknowledgment control message reception	end of waiting inside the successful answer primitive call, with transition into state "outside of protocol"
	"result?" control message reception	answer message retransmission
	any other message reception	ignore
	time out	end of waiting inside the answer primitive call, with transition into state "outside of protocol"

REFERENCES

[1] Birrell, A. D., Nelson, B. J., "Implementing Remote Procedure Calls", *ACM Transactions on Computer Systems*, 2, (1984) 39-59.

[2] Cheriton, D. R., Zwaenepoel, W., "The distributed V kernel and its performance for diskless workstations", *ACM Operating System Review*, 17 (1983) 129-140.

[3] Cheriton, D. R., "The V kernel: A software base for distributed systems", *IEEE Software*, 1984, 19-42.

[4] Cheriton, D. R., "The V distributed system", *Communications of the ACM*, 31 (1988) 314-333.

[5] Mullender, S. J., "Principles of distributed operating system design", Ph.D. thesis,Vrije Universiteit te Amsterdam, 1985 .

[6] Schroeder, M. D., and Burrows, M., "Performance of Firefly RPC", *ACM Transactions of Computer Systems* 8 (1990) 1-17.

[7] Tanenbaum, A. S., and Van Renesse, R., "Distributed operating systems", *Computing Surveys* 17 (1985) 419--470.

[8] Tanenbaum, A. S., Van Renesse, R., Van Staveren, H., Sharp, G., Mullender, S. J., Jensen, A., and Van Russum, G., "Experiences with the AMOEBA distributed operating system", *Communication of the ACM* 33 (1990) 46-63.

[9] Zgonjanin, D., "Comparison of interprocess communication mechanisms within distributed computer systems", Master thesis, Faculty of technical sciences, University of Novi Sad, 1994.

[10] Zgonjanin, D., Hajduković, M., Perišić B., "Improvement of client server model process cooperation by application of the active side principle", *Workshop on information technologies, system control and system management*, University of Novi Sad, May 1994.