GRPC: A Communication Cooperation Mechanism in Distributed Systems

Xingwei Wang, Hong Zhao and Jiakeng Zhu

Northeast University of Technology

Shenyang, 110006 P.R.China

Abstract

RPC (Remote Procedure Call) is very popular in distributed application design. However, most existing RPC systems have some defects on parallelism and reliability and transparency, they can only support point-to-point and(or) broadcast communication. We try to bring the process group concept into RPC systems and present a new kind of communication cooperation mechanism supporting the cooperation among distributed entities in distributed systems. We call it GRPC(Group Remote Procedure Call) mechanism. According to different application background, the related GRPC mechanism are named *lookup, function-convergence* and *update* GRPC. The prototype systems of these three specific classes of GRPC mechanisms have been implemented in the local area network which consists of SUN 4 workstations. The performance evaluation on these prototype systems has been made. Practice has shown: the GRPC mechanism can not only improve the parallelism, reliability and transparency of RPC systems, but also enable RPC systems to support group communication.

Key Words: RPC GRPC lookup-RPC function-convergence-GRPC update-GRPC

1. Introduction

As a new kind of interprocess communication mechanism in distributed computer systems and computer networks, RPC (Remote Procedure Call) is very popular in distributed application design because of its simplicity and transparency. Many RPC systems, such as SUN RPC and XEROX RPC, have been put into use and widely accepted by the distributed application designers. This has presented a brilliant development future for Interprocess Communication Mechanism. However, RPC has many new characteristics which are different from local procedure call. In RPC mechanism, the caller and callee of RPC are different processes located on different computers (or the same computer). The calling parameters and the return results are transferred between the caller and the callee in messages through the underlying network routines. If something goes wrong with the process, computer or communication mechanism, RPC will probably commit failure. Further, the failure of one calling side of RPC does not affect the survival of the

other side. This is the so-called *independent failure* scheme in RPC mechanism. For example, we assume that the caller has successfully started the callee in one RPC call. Just at this very moment, the caller has crashed due to process or computer disaster. The RPC call has failed, but the callee process is running continuously without knowning the caller process has failed and become the *orphan* process. There is no doubt that the *independent failure* scheme has greatly affected the reliability and transparency of RPC mechanism. In the meanwhile, since the caller and the callee of the RPC do not share the same address space, the RPC mechanism possesses the potential parallelism.

Unfortunately, most existing RPC systems have some defects on the reliability, transparency and parallelism. Usually, they can not support the detection and handling of the *orphan* process; they can only support the point-to-point and/or broadcast communication, and they can not support one-to-many communication. How to enhance the reliability and transparency of the RPC mechanism and how to explore the potential parallelism of the RPC mechanism have become very important and urgent in the RPC research realms. On the other hand, as the operating-system-level abstraction mechanisms, the process group and group communication mechanisms have been applied to the distributed application design widely. They can surely enhance the system fault-tolerance ability, improve the system reliability and transparency, and simplify the distributed application design.

Thus, we think: if the process group concept is brought into RPC systems, the shortcomings of the existing RPC systems can be overcome by the advantages of the group mechanisms. This surely can not only improve the reliability, transparency and parallelism of RPC systems, but also enable RPC systems to support group communication. As a new kind of communication cooperation mechanism supporting the cooperation among distributed entities in distributed systems, we call it GRPC (Group Remote Procedure Call) mechanism. According to different application backgrounds, the related GRPC mechanism are named *lookup, function-convergence* and *update* GRPC.

The remains of this paper are arranged as following: In section 2, related work with GRPC mechanism is introduced. In section 3, we will introduce distributed application background of different classes of GRPC mechanisms. In section 4, the design of the three different classes of GRPC mechanisms is described; In section 5, the implementation of prototype systems of the three different classes of GRPC mechanisms is presented. In section 6, the performance evaluation on the prototype systems of the three specific classes of GRPC mechanisms is given briefly. In section 7, the future work on GRPC mechanisms is discussed. Finally, we draw some conclusions in section 8.

2. Related Work

We can regard GRPC mechanism as the combination of RPC systems and process group concept. Here, we present several typical related RPC systems and group mechanisms as following, from which we have got profitable hint.

- Replicated Procedure Call [9]

Replicated procedure call mechanism is the combination of the RPC mechanism and the module replication. As one efficient mechanism which can be used to construct highly available distributed application programs, the replicated procedure call can greatly enhance the system fault tolerance ability. In this mechanism, the set of the module replicas is called to be *troupe*. When the distributed application programs are constructed, *troupe* can be used just as the normal procedure call. When failure occurs, even if

there is one player survived in the *troupe*, the distributed application program can be run continuously. [9] Also gives the semantics of the replicated procedure call and the definition of the *troupe* and describes implementation methods.

• Fault Tolerant RPC Mechanism [5]

In this mechanism, both the caller and the callee can be the replicated procedures, which are in the linear order and called cluster. When the fault tolerant RPC occurs, the remote procedure call request is sent to the first replica in the callee cluster, and then is handed to the other replicas sequentially. The first normal replica in the caller cluster returns its executing result to the caller as the fault tolerant RPC result. [5] discusses the creation, reconstruction and integrity maintenance of the cluster in details. [5] has implemented the mechanism on the basis of the existing SUN RPC systems and has described its performance evaluation in the local area network which consists of SUN workstations.

• Reliable Group Communication Mechanism [7]

This kind of reliable group communication mechanism can not only ensure the atomicity of the event delivery, but also ensure the ordering of the event delivery. Therefore, it can simplify the distributed application design.

• ABCAST, CBCAST and GBCAST Broadcast Communication Primitives [2]

As the reliable group communication mechanism in the ISIS systems, these communication primitives have provided the atomicity and ordering of event delivery and the group view integrity maintenance efficiently. As successful communication cooperation mechanisms in distributed systems, these systems have presented many useful new concepts and implementation methods on the RPC and group mechanisms. This has surely urged the formation of the GRPC concept.

3. The Application Background of GRPC Mechanism

According to the current development situation and application background of distributed computer systems, we further present three specific classes of GRPC mechanisms which are oriented to different distributed applications.

3.1 *lookup* GRPC mechanism

The design goal of this class of GRPC mechanism is to enhance the reliability and availability of the RPC systems and to improve system response time. For this specific class of GRPC mechanism, when a client makes one GRPC call, it will invoke all members of the called remote server group (called RSG below) simultaneously. If there is one success reply returned by one member of the called RSG, this GRPC call will return successfully at once. Only when all members of the called RSG have declared their failures, will the GRPC call return with failure. When it is applied to fully replicated RSG, its application object belongs to nondeterministic data and operation homogeneous group. It is used to enhance the system availability. When this specific class of GRPC mechanism is applied to partial replicated or partitional RSG, its application object belongs to nondeterministic data heterogeneous and operation homogeneous group. It is used to improve the execution parallelism so as to shorten the system response time. When it is used to multi-version programming, its application object belongs to nondeterministic data homogeneous and operation heterogeneous group. It is used to improve the execution parallelism and to enhance the

system fault-tolerance ability. Because the application object of this specific class of GRPC belongs to *stateless* RSG, it needs not support the maintenance of group view integrity. It also needs not support the atomicity and ordering of event delivery.

3.2 *function-convergence* GRPC mechanism

In real distributed environments, many distributed entities are related with each other in their functions. We can regard them as a single logical entity with complete functions so that we can provide one simple and uniform service interface to the external environments. This can simplify the distributed application design. For example, a set of data may need several different processing (such as sum, multiply, sort, encipher, etc) in many cases. If we use the conventional RPC mechanism (point-to-point RPC), the above processing functions will only be called sequentially. This will surely prolong the client-wait-time and suppress the parallel processing ability in distributed systems. If we fully exploit the existence of many autonomous processors in distributed systems and disperse the above processing functions in different autonomous processors, these process functions will surely form one RSG which is convergence-in-function, dispersion-in-location and uniformity-in-logic.

When we use this specific class of GRPC mechanism, we can not only invoke many remote servers to handle data simultaneously so that we can improve the parallel degree of the execution, but also shorten the client-wait-time and enhance the friendness of system user interface, certainly improve the transparency of RPC systems. At the same time, the load-balance in distributed systems will profit from this. According to the application background, this specific class of GRPC mechanism should support the integrity maintenance of group view and the atomicity of event delivery, but it needs not support the event ordering. The application object belongs to the *paradeterministic* data homogeneous and operation heterogeneous group.

3.3 *update* GRPC mechanism

The application background of this specific class of GRPC mechanism is to support the *update* operation of fully replicated systems so that the system reliability can be enhanced greatly. Usually, only when the client get the success reply from every member of the called RSG, will the GRPC call return successfully. Or, if any one member of the called RSG has declared its failure, the GRPC call will return its failure at once. Because this class of the GRPC mechanism is oriented to the *stateful* RSG, it should maintain the integrity of group view, support the atomicity and ordering of event delivery. Needless to say, the application object of this specific class of GRPC mechanism belongs to deterministic data and operation homogeneous group. It can surely simplify the design of distributed database and distributed processing algorithm. For *lookup* and *update* GRPC mechanisms, the function of each member of the RSG should be the same. For *function-convergence* GRPC mechanism, the function of each member of RSG can and should be different.

4. Design of GRPC Mechanisms

All three different classes of GRPC mechanisms mentioned in the section 3 have the same logical structure as the figure 1.

In this section, we describe the GRPC mechanism design in four aspects: normal GRPC calling process, GRPC result handling, GRPC group management and the atomicity and ordering of event delivery

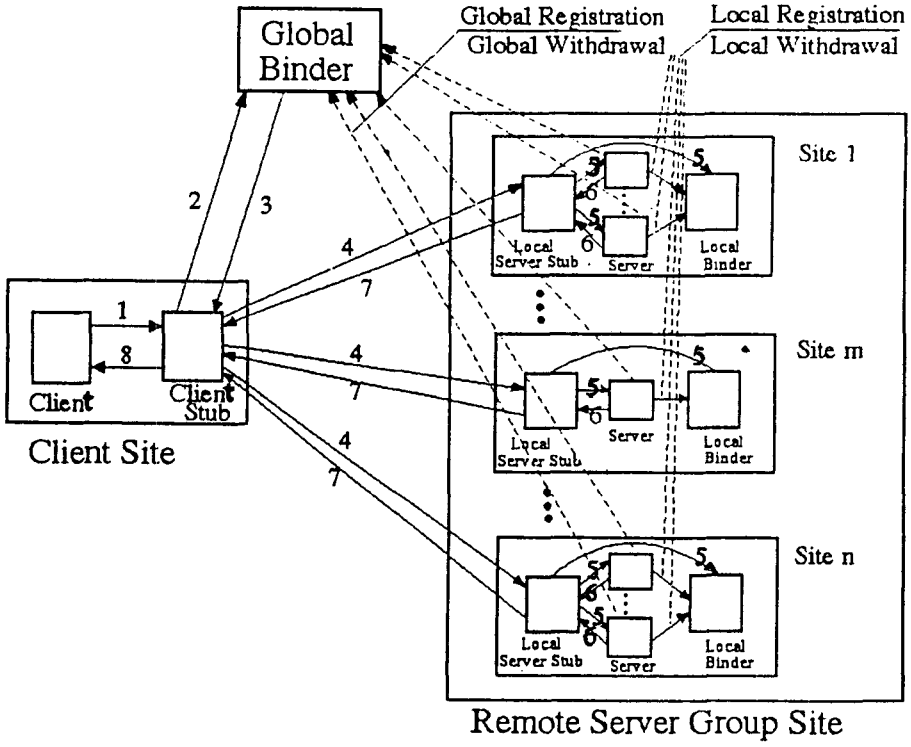

Figure 1. the logical structure of GRPC mechanism

4.1 Normal GRPC Calling Process

Normally, when a client makes one GRPC call (step 1 in figure 1), the client stub will collect the necessary calling parameters (for example, the name of the called RSG, etc), package them up and put them into XDR (eXternal Data Representation) format, and then send them to the global binder in one or more network messages (step 2 in figure 1). After it has received the request message, the global binder will translate this received request message into the local data representation format, make certain authentication on it, marshal membership information of the called RSG and put it into XDR format, and then send it to the corresponding client stub in one or more network messages (step 3 in figure 1). As soon as it receives the membership information of the called RSG, the client stub will marshal the related calling parameters (including data which is to be processed, etc) and put them into XDR format, and then send this remote service request message to each of the local server stubs of the called group according to the received group membership information which has been put into local data representation format (step 4 in figure 1). After it receives the corresponding remote service request message, each of the corresponding local server stubs of the called group will translate the received request message into local data representation format and interpret the related calling parameters, and then call the corresponding specific server procedure with the help of the local binder (step 5 in figure 1). When it is finished, the server procedure returns to its stub, returning whatever its return values are (step 6 in figure 1). The local server stub

marshals the returned execution result, puts it into XDR format, and then sends it to the corresponding client stub (step 7 in figure 1). The client stub collects all returned execution results, handles them to form the final GRPC call result, and returns this final GRPC call result to the client in the local data representation format (step 8 in figure 1). One normal GRPC call is finished. Because the data exchange among the calling and called sites and global binder is in XDR format, the GRPC mechanism can be used in heterogeneous environments.

4.2 GRPC Result Handling

Transparency is very important to the GRPC mechanism. In fact, it must be masked to the client that multiple replys would be generated by the called RSG. This requires that the client stub should have the *result handler* function so that the transparency of GRPC mechanism can be ensured. For the *lookup* GRPC mechanism, as soon as the result handler of the client stub gets one success reply from any member of the called RSG, the client stub will return successful GRPC call result to the client at once. This declares that the GRPC call is finished successfully. All other replys which comes lately will be discarded as *garbage* by GRPC systems. Only when the result handler concludes that all members of the called RSG have failed, will the client stub return the failed GRPC call sign to the client. And this declares that the GRPC call is finished with failure. For the *update* GRPC mechanism, only when the result handler of the client stub confirms that it has already got success replys from all members of the called RSG, will the client stub return the successful GRPC call result to the client and this declares that the GRPC call is finished with success. As long as the result handler concludes that one member of the called RSG has failed, the client stub will return the failed GRPC call result to the client at once and this declares that the GRPC call is finished with failure; all other replys which come lately will be discarded as *garbage* by GRPC systems.

For *function-convergence* GRPC mechanism, the result handler of the client stub is more special and more complicated than the *lookup* and *update* GRPC mechanisms. The *function-convergence* GRPC mechanism requires that the result message returned by each member of the called RSG should have *self-description* ability, i.e, the result message should have such header that it can describe the specific function of this specific member. The result handler of the client stub should not only judge the execution status (success or failure) of each member of the called remote server group by its returned result message, but also collect all returned result messages to form the GRPC call result *text* by their headers. At the same time, the result handler records the information about those failed members of the called RSG in this GRPC call. Such information is regarded as the GRPC call result *appendix* and is bound to the GRPC call result *text* back. Doing as such, the client not only can get the useful processing results from the GRPC call result *text,* but also can call the specific failed members lately with conventional RPC separately according to the GRPC call result *appendix* so that the nonnecessary overhead which is caused by using GRPC again is avoided (this nonnecessary overhead is actually caused by calling those members of the called RSG which have returned their processing results to the client in the last GRPC call successfully).

In fact, the result handler of the client stub can be regarded as one specific *broker* which is responsible for processing result messages returned by the called RSG.

4.3 GRPC Group Management

It is two level binding architecture -- global binding and local binding that the GRPC mechanism adopts (see figure 1). The global binder is responsible for the management and maintenance of the RSGs. The local binder is responsible for the management and maintenance of local servers -- those members of the RSG which are under control of the local binder. When any member of the RSG starts up, it will register itself on the local binder first, and then on the global binder. The *joining group* action is finished as such. After it joins the RSG, this specific RSG member is ready to accept the corresponding remote service requests. It can stop providing its service to the external world with the *withdrawing from group* action. When it withdraws from the RSG, this specific member will cancel itself from the global binder first, and then from the local binder. The *withdrawing from group* action is finished as such. The dynamic enlargement and shrinkage of the RSG is achieved in such manner.

We also stipulate rules as following: when the RSG starts up, the first successful *joining group* action is implicitly defined to be *creating group* operation. And the successful *withdrawing from group* action of the last member of the RSG is implicitly defined to be *destroying group* operation. Needless to say, all *creating group, destroying group, joining group* and *withdrawing from group* actions must be authenticated in proper manner. The integrity maintenance requirements of group view of these three specific classes of GRPC mechanisms are different because of their different application background. The *lookup* GRPC mechanism doesn't require the group view integrity maintenance, but the *function-convergence* and *update* GRPC mechanisms require it. When a client makes one *function-convergence* or *update* GRPC call, if the client stub hasn't received the reply from one member of RSG group and doubt it failed due to timeout, the client stub will express such doubt to the global binder and ask it to relay the corresponding remote service request message to the specific local server stub of that doubted member of the called RSG. As soon as it receives this specific doubt message, the global binder will relay that specific remote service request message to the specific local server stub of that doubted member and wait its reply. This time, it is as if that the global binder build a *bridge* between the client and the doubted member of the called RSG. If the global binder receives the reply message returned by that doubted member in fixed time interval, the global binder will relay this reply message to the corresponding client stub and tell the client stub the existence and liveness of that doubted" member. If the global binder doesn't receive the reply message from that doubted member due to timeout, the global binder will conclude that this specific member of the called RSG is actually failed and tell this conclusion to the corresponding client stub, and then delete it from the corresponding RSG. If several client stubs express their doubts of the same specific member of the called RSG to the global binder at the same time, the global binder will merge these doubts and complete the above work. It is this kind of *bridge* or *relay* function of the global binder that ensures the group view integrity maintenance to a certain degree. If this specific member of the RSG restores from its failure, it should join the RSG and provide service to the client again.

4.4 The Atomicity and Ordering of Event Delivery

The three different classes of GRPC mechanisms have different requirements on the atomicity and ordering of event delivery on the side of the RSG due to their different distributed application backgrounds. The *lookup* GRPC mechanism requires neither atomicity nor ordering of event delivery. The *function-convergence* GRPC mechanism only requires the atomic delivery of events. The *update* GRPC

mechanism requires both atomicity and ordering of event delivery. The *bridge* function of the global binder in *function-convergence* and *update* GRPC mechanisms have ensured the atomicity of event delivery in these two specific classes of GRPC mechanisms to a certain degree.

From figure 1, we can see that the client stub must get the membership information of the called RSG from the global binder first in order to complete the GRPC call actually. According to this, we think: for the *update* GRPC mechanism, the event ordering on the side of the RSG should be achieved by each local server stub of the called group with the participation of the global binder. Concrete method is described as following: when the global binder returns the membership information of the called RSG to the corresponding client stub, the global binder will also allocate the unique transaction identifier for the GRPC call which is a monotonous sequence number and *paggyback* it at the same time. The client stub then build the remote service request message according to this unique transaction identifier. When the remote service request message has been received by the corresponding local server stubs of the called group, each of the corresponding local server stubs will schedule and execute the specific called server procedure. The scheduling of the called server procedure is based on the monotonous sequential order of the transaction identifier of the received remote service request message and obeys such rules -- *discarding the out-of-date remote service request message, queuing the remote service request message which comes early, executing the exactly remote service request message*. This ensures the ordering of event delivery on the side of the called RSG.

In fact, the transaction identifier is as if a logical timestamp. If one local server stub cannot ensure the ordering of the event delivery due to the permanent loss of one event, the local server stub will commit *suicide* to avoid CPU time waste.

5. Implementation of Prototype Systems of GRPC Mechanism

According to the hardware and software configuration in our research laboratory, we have implemented prototype systems of these three specific classes of GRPC mechanisms in the local area network which consists of SUN 4/65 workstations and SLCs. The prototype systems of these three specific classes of GRPC mechanisms are based on the 4.3 BSD operating system, UDP/IP protocol and SOCKET mechanism. Their user interfaces are equivalent to the middle layer of SUN RPC user interface and are defined in C programming language as following strictly.

5.1 *lookup* and *update* GRPC mechanism

```
callgrpc(groupname, inproc, in, outproc, out)
char *groupname;
xdrproc_t inproc, outproc;
caddr_t in, out;
registergrpc(groupname, prognum, versnum, procnum, procname, inproc, outproc)
char *groupname;
u_long prognum, versnum, procnum;
char *(*procname)();
xdrproc_t inproc, outproc;
unregistergrpc(groupname, prognum, versnum, procnum, procname, inproc, outproc)
char *groupname;
u_long prognum, versnum, procnum;
```

*char *(*procname)();*
xdrproc_t inproc, outproc;
svcgrun()
gbinder()

The calling parameters are defined as such: *groupname* is the name of the called RSG; *inproc* and *outproc* are the corresponding XDR routines which are used to complete the translation between the local data representation format and the XDR format of the input parameter and the output result of the RSG member separately; *in* and *out* are the addresses of the input parameter and the output result separately; *prognum, versnum, procnum* and *procname* are the corresponding program number, version number, procedure number and procedure name of the RSG member. Here, we stipulate that the groupname parameter should not exceed 14 characters.

5.2 *function-convergence* GRPC mechanism

callgrpc(groupname, inproc, in, outplst, outlst)
*char *groupname;*
xdrproc_t inproc;
caddr_t in;
*struct result_hd *outplst;*
*char **outlst;*
registergrpc(groupname, prognum, versnum, procnum, procname, inproc, outproc, descript)
*char *groupname;*
u_long prognum, versnum, procnum;
*char *(*procname)();*
xdrproc_t inproc, outproc;
*char *descript;*
unregistergrpc(groupname, prognum, versnum, procnum, procname, inproc, outproc)
*char *groupname;*
u_long prognum, versnum, procnum;
*char *(*procname)();*
xdrproc_t inproc, outproc;
*char *descript;*
svcgrun()
gbinder()

The calling parameters are defined as such: *groupname* is the name of the called RSG; *inproc* and *outproc* are the corresponding XDR routines which are used to complete the translation between the local data representation format and the XDR format of the input parameter and the output result of the RSG member separately; *in* is the address of the input parameter; *outlst* is the list which will contain GRPC call result; *outplst* is the list of the result_hd structure, which the client stub uses to handle all sorts of results returned by the called RSG; *prognum, versnum, procnum, procname* and *descript* are the corresponding program number, version number, procedure number, procedure name and concise function description of the RSG member. Here, we stipulate that the groupname parameter should not exceed 14 characters and the descript parameter should not exceed 30 characters.

5.3 The functions of the mechanism

For every prototype system of these three GRPC mechanisms, the system routine callgrpc() is used to call the RSG actually; registergrpc() is used to complete the *joining group* or *creating group* action; unregistergrpc() is used to complete the *withdrawing-from-group* or action; svcgrun() is used to manage and schedule the local server procedures by the local server stub on the side of the RSG; gbinder() is responsible for the management and maintenance of RSGs, so it completes all necessary functions which the global binder in each specific class of GRPC mechanism should have. For the *lookup* and *function-convergence* GRPC mechanism, they provide *at least once* semantics to the client; for the *update* GRPC mechanism, when the GRPC call returns with failure, the *update* GRPC mechanism provides *at most once* semantics to the client; when the GRPC call returns with success, the *update* GRPC mechanism provides the *exactly once* semantics to the client.

6. Performance Evaluation

After we implemented prototype systems of the three specific GRPC mechanism, we have decided to make performance evaluation on them. We have made very large amount of performance tests on these prototype GRPC systems and the existing SUN RPC systems. We have got large amount of test data with these GRPC prototype systems and SUN RPC system. After we have carefully analyzed and computed test data, we make performance evaluation on these three specific classes of GRPC mechanisms. Limited to space, we only present the basic conclusions of the performance evaluation as following. When the network traffics is normal and the RSG member number is not very large and the remote procedure service time is not very short, the basic conclusion is surely correct. For the *lookup* and *update* GRPC mechanisms, we define *reliability factor* as the number of members which are located in different computers in the same RSG. For the *function-convergence* GRPC mechanism, we define Tg as the GRPC call time when we call the RSG using GRPC mechanism; and we define Tr as the sum of time when we call each member of the above RSG sequentially; we define the *speedup rate* as Tr/Tg.

- *lookup* GRPC Mechanism

1). With the same degree of the reliability, the GRPC overhead increases by 4% or less than SUN RPC.

2). When the *reliability factor* increases by 1, the GRPC overhead increases by 0.1% or less; the GRPC overhead can possibly reduce sometimes (Refer to 2).

3). When the remote procedure service time increases, the performance of GRPC mechanism improves.

- *function-convergence* GRPC mechanism

1). For the single member RSG, the *speedup rate* is not less than 0.97.

2). When the RSG member number increases by 1, the *speedup rate* increases by 0.4 or more.

3). When the remote procedure service time increases, the performance of GRPC mechanism improves.

- *update* GRPC mechanism

1). With the same degree of the reliability, the GRPC overhead increases by 4% or less than SUN RPC.

2). When the *reliability factor* increases by 1, the GRPC overhead increases by 0.1% or less.

3). When the remote procedure service time increases, the performance of GRPC mechanism improves.

7. Further Work on the GRPC Systems

The implemented prototype systems of these three specific classes of GRPC mechanisms are not perfect and should be improved further. Further work on these prototype systems should be done in the near future.

• Developing GRPC language and its complier GRPCGEN which is similar to the SUN RPC language and its complier RPCGEN.

At present, the user interface of implemented GRPC prototype systems is equivalent to the middle layer of SUN RPC user interface. This requires that the distributed application designers understand the implementation mechanism of GRPC prototype systems. Needless to say, this handicaps the popularity of GRPC mechanism to a certain degree. If we develop the GRPC language and its compiler GRPCGEN, we will mask the GRPC prototype system implementation mechanism to the client and can make the GRPC client stub and server stub and parameter handling routine generate automatically. This will surely enhance the transparency of GRPC mechanism and promote the popularity of GRPC system.

• Enhancing the robustness of global binder in GRPC prototype systems.

In GRPC mechanism, the global binder is responsible for group management and maintenance; its crash is a disaster to the GRPC prototype systems. As a temporary measure, we advice that the global binder should run on the most *robust* machine in the network. As a permanent goal, we advice that the global binder should also be replicated in group manner. These replicated global binders can be organized in *coordinator/cohorts* scheme or in cluster scheme and form a single logical global binder. Doing as such, we can enhance the robustness of the global binder and GRPC prototype systems without any change of the logical structure of GRPC mechanism.

• Handling the *orphan crowd* processes in GRPC mechanism.

In GRPC mechanism, if the client crashes during the calling process, it is possible to generate *orphan crowd* processes. Based on the successful experience of *orphan* process handling in conventional RPC mechanism and referring to the actual situation of GRPC mechanism, we can surely find the correct handling method of *orphan crowd* processes in the GRPC mechanism.

• Implementing the callback mechanism in GRPC prototype systems.

In real distributed environments, distributed entities usually have both client and server roles. Therefore, the GRPC prototype systems should support the callback mechanism and become symmetrical GRPC mechanism.

8. Conclusions

After we have implemented prototype systems of these three specific classes of GRPC mechanisms, we do a large amount of tests on these GRPC prototype systems and SUN RPC systems. Experiments have shown that the performance of these prototype systems is good and the expected goal of research and development of the GRPC mechanism has been achieved.

Theoretically, we do certain profitable work on the logical model and implementation mechanism of GRPC. Practically, we have implemented prototype systems of three different classes of GRPC mechanisms which support the cooperation between distributed entities in distributed systems. We have improved the parallelism, reliability and transparency of SUN RPC systems. We have implemented group communication mechanism which SUN RPC systems cannot support. Our GRPC prototype systems can support the dynamic enlargement and shrinkage of group. We have provided users with simpler distributed programming interface than SUN RPC, so that distributed application design will be simplified if using GRPC. The prototype systems of these three specific classes of GRPC mechanisms are compatible with SUN RPC systems.

Practice has shown: as a successful research on improvement of parallelism, reliability and transparency of RPC systems and as a useful endeavor on development of the communication cooperation mechanism in distributed systems, GRPC mechanism enjoys higher research value. We believe that more valuable, more creative, more practical and newer communication cooperation mechanism will be developed in the near future, which will surely push the research and development of distributed systems forward.

REFERENCES

[1] Andrew D. Birrell and Bruce Jay Nelson, Implementing Remote Procedure Calls, *ACM Transactions on Computer System,* Vol.2, No.1, Feb. 1984. p39-59.

[2] Kiam S. Yap, Pankaj and Satish Tripathi, Fault Tolerant Remote Procedure Call, *Proceeding of 8th International Conference on Distributed Computing System,* Jun. 1988. p48-54.

[3] S. Navaratnam, S. Chanson and G. Neufeld, Reliable Group Communication in Distributed Computing System. Jun. 1988. p439-466.

[4] Luping Ling, Samuel T. Channon and Gerald W. Neufeld, Process Group and Group Communications: Classifications and Requirements, *COMPUTER.* Feb. 1990. p55-66.

[5] Sape J. Mullender. Distributed Systems, *Addison-Wesley Publications,* 1989.

[6] David R. Chriton and Willy Zwaenepoal, Distributed Process Groups in the V kernel, *ACM Transactions on Computer Systems,* Vol.3, No.2, MAY 1985, P77-107.

[7] K. P. Birman and T. Joseph, Reliable Communication in the presence of Failures, *ACM Transactions on Computer Systems,* Vol.5, No.1, Feb 1987, P47-76.

[8] Zhao Hong and Wayne McCoy, An Association Model for Distributed Application Object in Distributed Systems, *ACM SIGOPS Operating System Review,* Oct 1990.

[9] Eric C. Cooper, Replicated Procedure Call, *ACM Operating Systems Review,* Vol.20, No.1, Jan 1986, P44-56.

[10] Network Programming, *SUN MicroSystems,* Revision A of 9 May, 1988.