Issues on the RPC Paradigms in Active Networks

Cheng-Zen Yang and Chien-Wen Chen
Department of Computer Engineering and Science
Yuan Ze University
Chungli, Taiwan, R.O.C.
E-mail: {czyang,antony}@syslab.cse.yzu.edu.tw

Abstract

The conventional Remote Procedure Call (RPC) model provides a distributed program execution model on an end-to-end client/server execution basis. In such a calling scenario, traditional RPC mechanisms are implemented under a basic assumption that the underlying network just provides a store-and-forward communication channel that will not modify the content of RPC packets. However, active networks provide another communication infrastructure that allows the intermediate active routers (nodes) to execute program codes and even to modify the packets. In this paper, three major issues will be addressed. The first issue is the binding mechanism in RPC invocation. The second issue is about the code execution. The third issue is about load balancing. In the end, an RPC paradigm for active networks is discussed for future development.

1. Introduction

The conventional Remote Procedure Call (RPC) model [2,3] provides a distributed program execution paradigm on an end-to-end client/server execution basis [5]. In this traditional paradigm, while a client program calls a remote procedure, the called procedure is executed by another process, usually on a remote server. Then the results are returned via the network to the requesting program. In such a calling scenario, traditional RPC mechanisms are implemented under the basic end-to-end argument [6] assumption that the underlying network just provides a store-and-forward communication channel that will not modify the content of RPC packets. Even in Java's RMI (Remote Method Invocation) [11] and XML-RPC [15], the calling scenarios are all based on the traditional end-to-end argument.

However, the emerging technology of active networks (AN) [5,10,12] provides another communication infrastructure that allows the intermediate active routers (nodes) to execute user-injected programs and even to modify the packets. The injected program code can be encapsulated in *active packets* [9,13,14], the packets containing program code, or downloaded from a specific *code server* [1]. Therefore, new network service protocols can be rapidly deployed. This new network infrastructure shows a possibility that the performance of traditional RPC calling mechanisms can be further improved by exploiting the benefits of active networks.

To exploit the high programmability of active networks in an active RPC mechanism design, several challenges will be confronted. In this paper, three major issues are addressed and discussed. The first issue is remote service binding in RPC invocation. If RPC service binding can be performed in the intermediate active nodes, RPC execution is highly adaptable to the contemporary environment. The active nodes will dynamically select a suitable remote server, with lightest load or fastest response time, and forward the RPC requests to the server. However, since the intermediate active nodes will perform the binding task, transparency needs to be considered in dynamic RPC service binding.

The second issue is about the performance optimization of code execution. Since active networks allow the intermediate active nodes execute user programs, remote RPC services should be performed in the proximate active node instead of the remote RPC server to minimize the communication delay and relieve the load of the remote server. In addition, the availability of RPC services is also highly increased. However, if the active node evicts the RPC code immediately after the end of the procedure is reached, reloading the code in next RPC calling will lengthen the RPC execution and burden the network load. To relieve the problem, a code cache needs to be incorporated. Cache management and RPC instantiation need to be further considered.

The third issue is about load balancing. Since

the workload of a remote RPC server can be distributed over the intermediate active nodes in active networks, whether a RPC task needs to be migrated and performed in a proximate active node should depend on the node's workload. In this paper, we discuss this load balancing issue and propose a tentative approach.

In this paper, an active RPC paradigm for active networks is also discussed for future development. Currently, we are now designing active RPC modules and implementing the proposed mechanisms on the basis of the ANTS toolkit [13,14].

The remainder of the paper is organized as follows. Section 2 briefly reviews the traditional RPC mechanisms and current development. Section 3 introduces the network model on which the proposed active RPC paradigm is built. Section 4 discusses three major design issues and depicts the approaches to facilitate active RPC. Finally, Section 5 concludes the paper and discusses the future work.

2. RPC and AN Background

To facilitate the cooperation between distributed processes, explicit message exchange plays an important role in many distributed systems. However, explicitly handling messages complicates the programming development of distributed tasks. To reduce the programming complexity, the RPC programming paradigm is proposed on the observation that procedure calls provide a more natural programming mechanism to let remote tasks be performed as in a single local computer [2,3]. In 1984, Birrell and Nelson introduced such an RPC mechanism in detail to show how to implement the RPC and solve the underlying subtle problems [2]. Figure 1 depicts a typical RPC calling scenario.

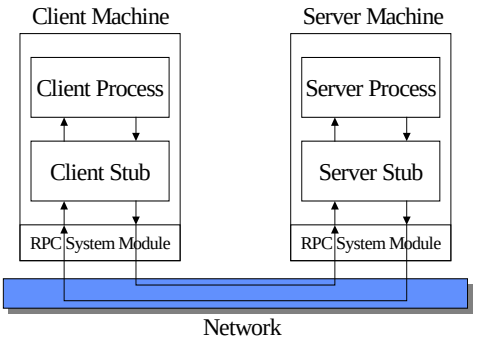

Figure 1: The traditional RPC paradigm.

The RPC mechanism is performed as follows. The client process calls the RPC procedures in a normal way as if the procedures are in a local library. The procedures in fact forward the requests to the underlying client *stub* module that is combined in the client process when the client program is compiled. Then the client stub builds the RPC messages and sends the messages to the remote server *stub* through the RPC OS module. After receiving the RPC request messages, the server stub unpacks the messages and dispatches them to the corresponding server procedures. The server procedure processes the request messages and sends the results back to the calling client process through the server stub and client stub.

In this typical RPC calling scenario, the network model follows the end-to-end programming model [6]. That is, the underlying network just provides a store-and-forward communication channel. The intermediate routers can only perform the computation work up to the network layer of the OSI seven layer model. However, the active network architecture shows a new perspective that the active routers can perform computations on the user data [5,10,12]. Figure 2 depicts a typical scenario of application specific processing in active networks. Currently, several active network environments have been proposed and implemented such as ANTS in MIT [13,14], PLAN in the University of Pennsylvania [1,4], and Netscript in Columbia University [16].

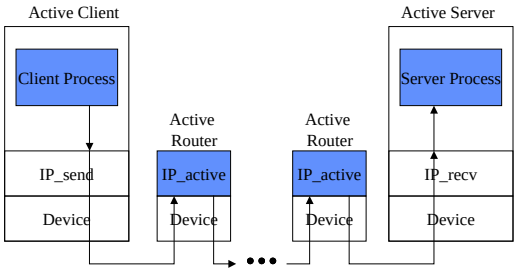

Figure 2: Application specific processing in active networks.

In active routers, customized programs are executed in each respective execution environment (EE). The customized program code can be injected from active packets/capsules, or downloaded from code servers. Therefore, new network services and protocols can be dynamically deployed with high flexibility.

Nowadays, although different technologies such as object-oriented technology and XML have been integrated with RPC mechanism design, current RPC development does not exploit the

benefits of the active networking architecture. For example, in Java's RMI [11] and XML-RPC [15], the calling scenarios are all based on the traditional end-to-end argument. However, if the procedure code of the remote RPC server can be downloaded to the proximate active router and executed in the EE on the active router, three major benefits can be achieved.

1. The RPC server's load can be shared with the active router. Because the server's load is distributed over the active routers, the RPC execution performance can be highly improved.
2. Code caching can be supported to reduce the communication latency in further RPC calls. If other nearby clients call these cached RPC procedures, their performance will be also improved because lengthy remote communications are avoided.
3. The availability of RPC services can be highly improved because the services are provided in the nearby active routers rather than the faraway RPC servers.

However, some issues need to be discussed in such an active RPC mechanism design. To achieve load balancing, high performance, and high availability, code caching is the key component in active RPC design. Code caching should dynamically adapt to the variations of the network environment.

3. Active RPC Model

Figure 3 depicts the proposed active RPC paradigm. In this paradigm, the architecture of client process and server process are basically same as in the traditional RPC architecture. However, the intermediate router now exploits the active networking technology.

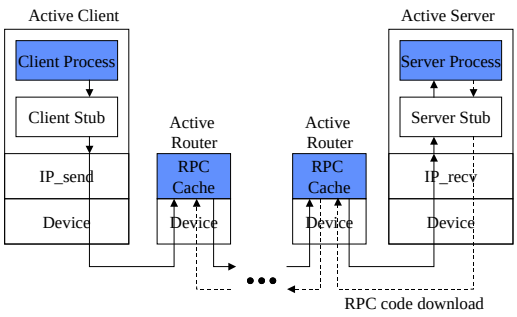

Figure 3: Active RPC paradigm in active networks.

In each active router, an RPC cache exists to cache the remote RPC procedure code. Besides the RPC cache, an active RPC monitor (ARM) operates on the active router to activate the cached RPC procedure. If the requested RPC procedure is not in the RPC cache, ARM will forward the RPC requests to the remote RPC server. The RPC server will perform the requested computations and send back the results with piggybacked RPC procedure code. However, if an active router between the requesting ARM and the remote RPC server along the forwarding path has the RPC procedure code, it will intercept the requests and perform the RPC server work. Therefore, the requests can be resolved as soon.

ARM also takes the responsibility for managing the RPC cache. When the amount of the available cache space is smaller than a predefined threshold, ARM performs replacements to evict least-recently used RPC code. When data in the active router are changed, ARM will reflect these changes back to the RPC server. Since there may be multiple RPC caches having the same RPC procedure code, data inconsistency is possible. When data inconsistency occurs, ARM performs resolving algorithms to control the consistency and invalidate or update remote inconsistent data.

To facilitate RPC monitoring in the active router, the client stub and server stub need to be extended. The client stub needs to provide authentication information to check the integrity of the cached RPC code. The information is also used by ARM to get the RPC code from the remote RPC server.

Since failures such as site crashes and network disconnections may occur, the server stub needs to handle the possible data inconsistency. Though a stateless server stub design can achieve high performance, non-idempotent RPC procedures may suffer from complicated failure handling. An AFS-like callback mechanism [8] could be used in the server stub design to make the caching and failure handling more efficient. To avoid data lost, the active router can store updated data in non-volatile memory first and remote them after the updates are committed in the RPC server.

4. Design Issues

To extensively exploit the high programmability of active networks in an active RPC mechanism design, challenges will be confronted. In this section, we only address and discuss three major issues. The first issue is how to perform remote service binding in active RPC invocation. The second issue is about how to

effectively optimize the RPC execution performance with RPC caches. The last issue is load balancing.

4.1 Dynamic Binding

Since ARM will intercept the RPC requests, RPC service binding is indeed performed on the active routers. Therefore, if the RPC service is replicated on another trusted server, ARM may actually bind the requests to this alternative server for some sake such as fault tolerance. Figure 4 depicts the dynamic binding.

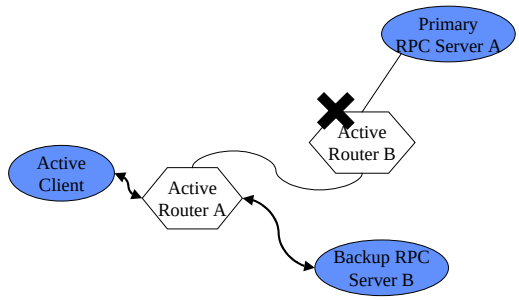

Figure 4: Dynamic binding in active RPC invocations.

In this figure, we assume there are two RPC servers. One is the primary RPC server A, and another is the backup RPC server B. When a client requests the RPC service, the request passes through two active routers. Now the active router B crashes. If dynamic binding is supported, the active router A will bind the RPC service to the backup RPC server B after it detects the crash failure of the active router B. Therefore, RPC service availability is highly increased.

To support dynamic binding, an integrity checking mechanism is needed in ARM and RPC stub design. ARM needs to check the code integrity to ensure the RPC procedure code in the backup RPC server B is identical to the original RPC procedure code in the primary server A. There are two possible approaches. The first is to introduce a service directory server (SDS). In SDS, the RPC services are registered in a database. Then the active router can perform dynamic binding by enquiring the SDS database. Another approach is to generate a unique authentication key for each RPC service when the stub code is compiled. Then the client stub sends this key during each RPC invocation. The active router will first try to bind the primary RPC server. If the primary RPC server is out of service, the active router then broadcast the requests with the key.

However, broadcasting may burden the network load but fail to find the backup RPC server. Therefore, the primary server may actively provide such information when the RPC code is first cached. The active router records the information for future use. Active clients may also explicitly specify several possible RPC servers when performing RPC invocations. The active router will try these candidates in RPC service binding.

4.2 RPC Cache Management

To achieve high performance, RPC cache management is the key component in the active RPC mechanism. On each active router, ARM takes responsibility for RPC cache management. In addition to RPC code replacement, ARM needs to maintain related information.

First of all, the RPC cache does not only need to cache the RPC procedure code, but also need to maintain the data processed by the code. If the allocated memory space of the data is not released after the RPC procedure returns, any modification on the data should be updated to the remote RPC server. Therefore, when another active router requests for the same RPC procedure code, it can see the previous updates.

Handling non-volatile data is important in many RPC applications such as NFS [7]. However, though immediately updating modifications minimize the inconsistence period, it hinders RPC from achieving high performance and burdens the network load. To relieve this problem, we first classify RPC procedures into two categories: the idempotent RPC procedures accessing only local variables, and the non-idempotent RPC procedures that may access non-volatile variables. Only the RPC procedure code in the second category needs to be considered. While an active router caches such a RPC procedure, it also gets a callback token from the remote RPC server. Any modification will not be immediately updated until the procedure finishes the execution, or the remote RPC server issues the callback. If the update is initiated, the update will be propagated to the remote RPC server along the caching path. If the updating path is different with the caching path, the RPC server will invalidate the corresponding parts in other active routers' caches.

ARM also needs to handle exceptions related to these persistent data. While an exception occurs, ARM will immediately propagate the exception to the RPC server and other active

routers. The RPC server will then make sure every active router that has cached the RPC code is accordingly notified.

4.3 Load Balancing

Load balancing is an important issue. If the active router is overloaded for performing RPC execution, its other tasks will suffer. Therefore, ARM needs to monitor the workload of the active router. When ARM detects that the load of the active router is saturated, it will forward the newly incoming RPC requests directly to next active router or the RPC server.

ARM needs also to monitor the resource consumption. When ARM detects that the amount of the available resource is lower than a threshold, it defers the incoming RPC requests for waiting for resource release. However, if the deferred period exceeds a timeout, the RPC requests will be forwarded to next active router or the RPC server.

5. Conclusions and Future Work

The emerging technology of active networks (AN) provides a new communication infrastructure that allows the intermediate active routers to execute user-injected programs and even to modify the packets. This new network infrastructure shows a possibility that the performance of traditional RPC calling mechanisms can be further improved by exploiting the benefits of active networks.

In this paper, we have discussed three major design issues on the active RPC paradigm exploiting active networking technology. if the procedure code of the remote RPC server can be downloaded to the proximate active router and executed in the EE on the active router, three major benefits can be achieved.

1. The RPC server's load can be shared with the active router. Because the server's load is distributed over the active routers, the RPC execution performance can be highly improved.
2. Code caching can be supported to reduce the communication latency in further RPC calls. If other nearby clients call these cached RPC procedures, their performance will be also improved because lengthy remote communications are avoided.
3. The availability of RPC services can be highly improved because the services are provided in the nearby active routers rather than the faraway RPC servers.

There are still many design challenges in developing the active RPC. For example, object-oriented method invocation is needed for distributed object-oriented computing. However, handling OO technology in RPC invocation is more complicated. Furthermore, security and RPC authentication need to be in-depth considered to avoid malicious operations.

In the future, we will first develop active RPC generation tools and implement a prototype system to demonstrate the feasibility. A comprehensive performance evaluation will be conducted to find the performance bottleneck. Building a more complicated experimental environment is expected in the next stage.

References

1. D. S. Alexander, W. A. Arbaugh, M. W. Hichs, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. "The SwitchWare Active Network Architecture." *IEEE Network*, pp. 29-36, May/June 1998.

2. A. D. Birrell and B. J. Nelson. "Implementing Remote Procedure Calls." *ACM Trans. on Computer Systems*, Vol. 2, No. 1, pp. 39-59, February 1984.

3. J. Bloomer. *Power Programming with RPC*. O'Reilly & Associates, Inc., 1991.

4. M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter and S. Nettles. "PLAN: A Packet Language for Active Networks." In *Proc. of Int'l Conf. Functional Programming*, pp. 86-93, 1998.

5. K. Psounis. "Active Networks: Applications, Security, Safety, And Architectures." *IEEE Communications Surveys*, pp. 2-16, First Quarter 1999.

6. J. H. Saltzer, D. P. Reed, and D. D. Clark. "End-To-End Arguments in System Design." *ACM Trans. on Computer Systems*, Vol. 2, No. 4, pp. 277-288, November 1984.

7. R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh and B. Lyon. "Design and

Implementation of the Sun Network File System." In Proc. of 1995 USENIX Conference, pp.119-130, June 1985.

8. M. Satyanarayanan. "Scalable, Secure, and Highly Available Distributed File Access." *IEEE Computer*, Vol. 23, No. 5, pp. 9-21, May 1990.

9. B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, And C. Partridge. "Smart Packets: Applying Active Networks to Network Management." *ACM Trans. on Computer Systems*, Vol. 18, No. 1, pp. 67-88, February 2000.

10. J. M. Smith, K. L. Calvert, S. L. Murphy, H. K. Orman, L. L. Peterson. "Activating Networks: A Progress Report." *IEEE Computer*, Vol. 32, No. 4, pp. 32-41, April 1999.
11. Sun Microsystems, Inc. Java Remote Method Invocation Specification. Available from http://www.sun.com/, October 1998.
12. D. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. "A Survey of Active Network Research." *IEEE Communications Magazine*, pp. 80-86, January 1997.
13. D. Wetherall. "Active Network Vision and Reality: Lessons from a Capsule-based System." In *Proc. of the 17th ACM Symp. on Operating Systems principles (SOSP'99)*, pp. 64-79, 1999.
14. D. Wetherall, J. Guttag, and D. Tennenhouse. "ANTS: Network Services without the Red Tape." *IEEE Computer*, Vol. 32, No. 4, pp. 32-41, April 1999.
15. "XML-RPC Specification (Userland)". Updated 16, October 1999. http://www.xmlrpc.com/spec.
16. Y. Yemini, and S. da Silva. "Towards Programmable Networks". In *Proc. of FIP/IEEE International Workshop on Distributed Systems: Operations and Management*, October 1996.