A COMPARISON OF TWO PARADIGMS FOR DISTRIBUTED COMPUTING

M. Frans Kaashoek
Andrew S. Tanenbaum
Kees Verstoep

ABSTRACT

This paper argues that a distributed system should support two communication paradigms: Remote Procedure Call (RPC) and group communication. The former captures point-to-point communication; the latter captures one-to-many communication. We demonstrate that group communication is an important paradigm by showing that a fault-tolerant directory service is much easier to implement with groups than with RPC and is also more efficient. The directory service exemplifies distributed services that provide high reliability and availability by replicating data.

1. Introduction

Distributed applications are often based on Remote Procedure Call (RPC) [Birrell and Nelson 1984]. Many distributed applications, however, can better be programmed with group communication. If an application is distributed over multiple processors, it often needs to send a message from one process to all other processes participating in the application. Unlike RPC, group communication allows a process to send a message from 1 source to *n* destinations. One can implement group communication by performing *n* RPCs, but this requires at least 2**n* messages, which is slow and wasteful of bandwidth. To allow efficient 1-to-*n* and 1-to-1 communication, the Amoeba distributed system supports both group communication and RPC [Tanenbaum et al. 1990; Mullender et al. 1990]. In this paper, we compare the two models of communication in the same context and give head-to-head performance comparisons.

To perform the comparison we will use one specific application: a fault-tolerant directory service. The directory service exemplifies the class of applications that provide a highly reliable and highly available service through replication of data. We compare two implementations of the directory service on the Amoeba distributed operating system: one based on RPC and one based on group communication. From the comparison we conclude that for this class of applications group communication is a more appropriate paradigm than RPC. The directory service based on group communication has a simpler design and has better performance.

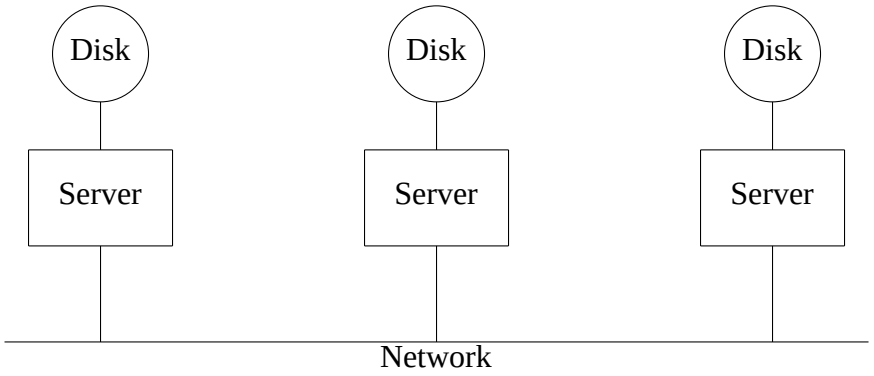

Fig. 1. Structure of the directory service.

The directory service is a vital service in the Amoeba distributed operating system [Van Renesse 1989]. It provides among other things a mapping from ASCII names to capabilities. A directory is basically a table with 2 columns: one storing the ASCII string and one storing the corresponding capability. Capabilities in Amoeba identify an object (e.g., a file). The set of capabilities a user possesses determines which objects he can access and which not. The directory service allows the users to store these capabilities under ASCII names to make life for them easier.

The directory service must be highly reliable and highly available. Users rely on the directory server to store capabilities without losing them and users must always be able to access their capabilities. To fulfill these demands the directory service replicates name-capability pairs on multiple machines, each with its own disk (see Fig 1). If one of the machines is unavailable, one of the other machines will be able to reply to a user's request. If one of the disks becomes unreadable, one of the other disks can be used to reply to a user's request. The key problem is to keep the replicas of a name-capability pair consistent in an efficient way. An update to a directory must be performed fast, because otherwise many applications will run less efficiently.

The architecture for a fully fault-tolerant directory service is more complex. Each directory server should be located on a separate electrical group (with its own fuse) and all the directory servers should be connected by multiple, redundant, networks. Because the Amoeba communication primitives are implemented on top of an internetworking protocol [Kaashoek et al. 1991], the latter requirement can be fulfilled. Although it could run on multiple networks without a single modification, the current implementation runs on a single network.

In Section 2 we discuss an implementation of the directory service using RPC. In Section 3 we discuss an implementation of the directory service using group communication. In Section 4 we will compare these two implementations and give performance measurements for both implementations. In Section 5 we will draw some conclusions based on the comparison.

2. Directory service based on RPC

The directory server based on RPC runs on two processors, each with its own disk. Thus, this implementation is only able to deal with processor failures, but not with network partitions. When a client performs a directory operation (e.g., append a new name-capability pair to a directory), a library routine performs an RPC with the directory service. One of the servers will get the request† and inspect the operation code stored in the request message. If it is a READ operation (i.e., an operation that does not change a directory), it performs the operation locally and sends a reply back. READ operations do not require any communication with the other directory server.

If the operation requested is a WRITE operation (i.e., an operation that possibly changes a directory), the two copies of the directory have to be updated in a consistent way. This is done using a two-phase commit protocol. The implementation of the protocol takes advantage of the fact that only two servers are involved. An implementation using 3 servers would require a much more expensive protocol.

The protocol works as follows. The server receiving the request informs the other server of its intentions. If this server is not performing an operation itself, it sends an acknowledgement, writes these intentions to disk, and executes the operation. After the original server has received the acknowledgement, it stores the intentions on disk, and performs the operation. At this point the operation is officially succeeded, and a reply is sent back to the client.

Thus, when the RPC returns successfully, the client knows that one server has stored the new directory on disk and that the other server has done the same or is about to the store the updated directory on disk. If either server crashes or either disk becomes unreadable, the client is still able to access his directories.

3. Directory service based on group communication

Amoeba supports process groups and the ability to send a message to all process in the group [Kaashoek and Tanenbaum 1991]. Messages to a group are sent *reliably* (i.e., there is automatic recovery from communication failures) and are *globally ordered*. The protocol implementing these semantics only requires 2 messages per reliable, globally ordered group message. When creating a group, a user can also specify a resilience degree *R*. With a resilience degree *R* Amoeba guarantees that even in the face of *R* simultaneous processor failures the message is still delivered at the remaining destinations in the same order. The costs for this protocol are $2 + R$ messages per fault-tolerant and globally ordered group message. Thus, by choosing *R*, users can trade performance against fault-tolerance.

The directory server based on group communication currently consists of 3 servers (though 4 or more is also possible), each with its own disk. They form a group with resilience degree of 2. To account for possible network partitions, at least two members should be up and in the same group. When one of the servers receives a re-

---

†The first time a client performs a RPC, the client kernel will broadcast a locate packet to find the location of the service. The server that responds first will receive the request. The client kernel caches the location of the service to avoid having to locate it on each request.

quest from a client, it inspects the operation code. READ operations are performed locally. As in the RPC implementation, READ operations do not require communicating with one of the other servers.

When a server receives a WRITE operation, it broadcasts the message to the group. All servers, including the sending server, will receive the broadcast in the same order and therefore execute all updates in the same order. After the original server has performed the update and stored the modified directory as a file on its disk, it sends a reply back to the client. Because group messages are sent atomically, the servers do not have to implement a two-phase commit protocol and do not have to store intentions on disk. Compared to the RPC implementation, the service based on group communication sends fewer messages and executes fewer disk operations.

When the client's RPC returns successfully, the user knows that one new copy of the directory is stored on disk and at least two other servers have received the update and have the stored the results on disk or will do shortly. Even if two processors crash, the client can still access his directories.

4. Comparison

Both implementations of the directory service run on the Amoeba operating system. The directory service based on RPC has been in use for several years. The directory service based on group communication has been in use for several months in an experimental setting and will be used as the default directory service shortly. Both directory services run on the same hardware: machines comparable to a Sun3/60 connected by 10 Mbit/s Ethernet and equipped with Wren IV SCSI disks.

Operation	Group	RPC	Sun NFS	Group +NVRAM
Append-delete	157	273	91	28
Tmp file	195	303	165	64

Fig. 2. Performance of 2 kind of directory operations for 3 different Amoeba implementations and for 1 UNIX implementation. All times in msec.

We have measured the performance of two kinds of operations. The results are shown in Fig. 2. The first experiment measures the time to append a new name-capability pair to a directory and delete it subsequently (e.g., appending and deleting a name for a temporary file). The second experiment measures the time to create a 4-byte file, register its capability with the directory service, look up the name, read the file back from the file service, and delete the name from the directory service. This corresponds with the use of a temporary file that is the output of the first phase of a compiler and then is used as an input file for the second phase. Thus, the first experiment measures only the directory service, while the second experiment measures both the directory and file service.

The implementation using group communication is 1.7 and 1.6 times more efficient than the one based on RPC, while the group directory service is triplicated and the

RPC implementation is only duplicated. The reason is the fact that the RPC implementation uses additional disk operations to store intentions safely.

For comparison reasons, we ran the same experiment using Sun NFS; the results are depicted in the third column. NFS does not provide any fault-tolerance or consistency (e.g., if another client has cached the directory, this copy will not be updated consistently when the original is changed). Compared to NFS, providing high reliability and availability costs a factor of 1.7 and 1.2 in performance.

The dominant cost in providing a fault-tolerant directory service is the cost for doing the disk operations. Therefore, we have implemented a third version of the directory service, which does not perform any disk operations in the critical path. Instead of directly storing modified directories on disk, this implementation stores the modifications to a directory in a 24Kbyte Non Volatile RAM (NVRAM). When the server is idle, it applies the modifications logged in NVRAM to the directories stored on disk. Because NVRAM is a reliable medium, this implementation provides the same degree of fault-tolerance as the other implementations, while the performance is much better.

Using NVRAM, some sequences of directory operations do not require any disk operations at all. Consider the use of */tmp*. A file written in */tmp* is often deleted shortly after it is used. If the append operation is still logged in NVRAM when the delete is performed, then both the append and the delete modifications to */tmp* can be removed from NVRAM without executing any disk operations at all.

Using group communication and NVRAM, the performance improvements for the first experiment are enormous (see fourth column in Fig. 2.). This implementation is 5.6 and 3 times more efficient than the pure group implementation and is 9.8 and 4.7 times more efficient than the RPC implementation. The implementation based on NVRAM is even faster than Sun NFS, which provides a lesser degree of fault-tolerance and has a lower availability.

5. Conclusion

In this paper, we claim that a distributed system should not only support RPC, but group communication as well. Group communication allows simpler and more efficient implementations of a large class of distributed applications. As an example to demonstrate the claim we looked in detail at the design and implementation of a fault-tolerant directory service. The directory service using group communication is not only easier to implement, but also more efficient.

Acknowledgements

Marc Wood provided comments on an earlier draft of this paper, which improved its content and presentation substantially.

References

Birrell, A. D. and Nelson, B. J., “Implementing Remote Procedure Call,” *ACM Trans. Comp. Syst.*, Vol. 2, No. 1, pp. 39-59, Feb. 1984.

Kaashoek, M. F. and Tanenbaum, A. S., “Group Communication in the Amoeba Distributed Operating System,” *Proc. The 11th Internatinal Conference on Distributed Computer Systems*, pp. 222-230, IEEE Computer Society, Arlington, TX, May 1991.

Kaashoek, M. F., Van Renesse, R., Van Staveren, H., and Tanenbaum, A. S., “FLIP: an Internetwork Protocol for Supporting Distributed Systems,” IR-251, Vrije Universiteit, Amsterdam, June 1991.

Mullender, S. J., Van Rossum, G., Tanenbaum, A. S., Van Renesse, R., and Van Staveren, H., “Amoeba: A Distributed Operating System for the 1990s,” *IEEE Computer*, Vol. 23, No. 5, pp. 44-53, May 1990.

Tanenbaum, A. S., Van Renesse, R., Van Staveren, H., Sharp, G., Mullender, S., Jansen, A., and Van Rossum, G., “Experiences with the Amoeba Distributed Operating System,” *Commun. ACM*, Vol. 33, No. 12, pp. 46-63, Dec. 1990.

Van Renesse, R., “The Functional Processing Model,” Vrije Universiteit, Amsterdam, 1989. (PhD. thesis)