Rapid-Response Replication: A Fault Tolerant Algorithm Based on Active Replication

Lingxia Liu, Jingbo Xia, Zhiqiang Ma, and Ruixin Li

The Telecommunication Engineering Institute,
Air Force Engineering University,
710077 Xi'an, China
lingxia_liu@tom.com

Abstract. To the wide area network oriented distributed computing, a slow service is equivalent to an unavailable service. It makes demands of effectively improving the performance of the replication algorithms without damaging the availability. Aim for improving the performance of the active replication algorithm, we propose a new replication algorithm named RRR (Rapid Response Replication). Its basic idea is: all replicas receive request, but only the fastest one sends back the response to the client after it handles the request. Its main advantages are: (1) In the algorithm, the response is sent back directly by the fastest replica after it handles the request; (2) The algorithm avoids agreement; (3) The algorithm avoids the redundant nested invocation problem (the problem may be arisen while using active replication).

Keywords: replication algorithm, performance, fault tolerant.

1 Introduction

Fault tolerance is the ability of an application to continue valid operation after the application, or part of it, fails in some way[1]. Replication[2] [3] is a widely used technique for providing high-availability and fault-tolerance of critical services.

After analyzing the Active Replication algorithm[2], we can find that agreement affects the performance of the algorithm from following aspects: (1) Agreement must be done after the slowest replica handling the request. The response time is decided by the slowest replica; (2) Agreement itself consumes time.

Aim for improving the performance of the active replication algorithm, we propose a new replication algorithm named RRR (Rapid Response Replication). Its basic idea is: all replicas receive request, but only the fastest one sends back the response to the client after it handles the request. Its main advantages are: (1) In the algorithm, the response is sent back directly by the fastest replica after it handles the request; (2) The algorithm avoids agreement; (3) The algorithm avoids the redundant nested invocation problem (the problem may be arisen while using active replication).

The rest of the paper is organized as follows. In Section 2, we set forth the algorithm. Next Section analyzes the performance of the algorithm. Finally, in Section 4, we conclude the paper.

Y. Shi et al. (Eds.): ICCS 2007, Part III, LNCS 4489, pp. 133–136, 2007.
© Springer-Verlag Berlin Heidelberg 2007

2 RRR Algorithm

We categorize the messages passing in the system into the following five types: C_Req (Client Request): a request message from the client to the service; C_Res (Client Response): a response message from the service to the client; S_Req (Service Request): a request message from one service to the other services; S_Res (Service Response): a response message from one service to the other services; State_Update: a state message from one service to the other services. After processing the message, the state of the service is set to the state in State_Update.

2.1 The Components for the Algorithm

The components for the algorithm are shown in Figure 1.

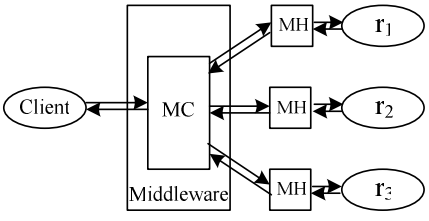

Fig. 1. The components for the algorithm

- Message coordinator (MC)

The MC component is located in the middleware. It assigns a unique identifier (ID) to each request and delivers them to RS. Once RS returns results, MC returns them to clients. It also coordinates RS to decide which replica is the fastest one.

MC assigns and attaches ID to every request message it handles. The generating rule is as follows: All the C_Req are identified with a unique ID; The ID of C_Res is the same as the ID of the corresponding C_Req; The S_Req is generated only when a service sends an invocation to another service. The ID is modified to ID (ID of the corresponding C_Req or S_Req) + URN (URN of the requested service); The ID of S_Res is the same as the ID of the corresponding S_Req; The ID of State_Update is the same as the ID of the corresponding C_Req or S_Req.

- Message handler (MH)

MH is co-located with each member of RS. It is placed in each replica to handler the messages that the replica receives and generates.

MH maintains a buffer for buffering the messages for every member in the group. All messages sending to the members are firstly buffered in the buffer. MH is responsible for scheduling the message in the buffer.

2.2 Overview of the Algorithm

In RRR algorithm, the request message is atomic multicast to every member of RS. The fastest member returns the response message and sends a State_Update to the other members in the group. It is the core idea of the algorithm.

Figure 2 shows a simple flow of the algorithm:

1. A client issues request C_Req to the replicated service.
2. The request is routed to MC (by the middleware). MC assigns a unique ID to C_Req and then multicasts it to the replicated service.
3. C_Req is intercepted and buffered by each MH. Then it waits for scheduling.
4. MH invokes the corresponding replica to handle the request.
5. The replica handles the request and returns response C_Res. C_Res is also intercepted by corresponding MH. MH decides whether the replica is the fastest one to finish handling the request. If the replica is the fastest one, MH sends back C_Res to the client through MC and sends State_Update to other members.

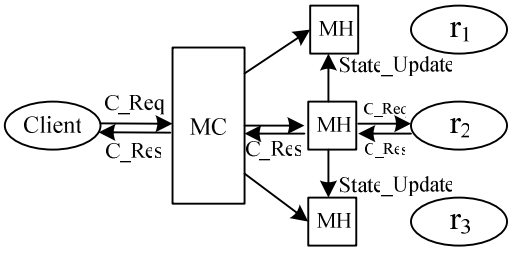

Fig. 2. A simple flow of RRR algorithm

Every member of the group decides whether to process the request or not on the system load and the processing speed of the request. The request doesn't need to be processed by every member and the response can be returned before every member finishes processing the request. They are the main differences from Active Replication.

MC maintains a hash table to ensure that there is only one member to be appointed as the fastest one (the members maybe finish processing the same request message at the same time). If the member can't decide whether it is the fastest one, MH will send an asking message to MC to confirm. MC returns the answer according to the record in the Hash table. Thus two types of messages are increased in the system, namely, Fastest_Query (the asking message from MH to MC) and Fastest_Response (the answer from MC to MH). The ID of the Fastest_Query and Fastest_Response is the same as the ID of the corresponding C_Req or S_Req. The date in the hash table is arranged as (address list, the fastest member's address, time). The address list includes the address of the members that ever sends out the Fastest_Query except the fastest one.

3 Performance Analyze

Experiments are designed to measure the response time of these algorithms. Our measurements were performed on 2.0GHz Intel Pentium4 PCs interconnected by a 100 Mb/sec switched. All machines run on a Windows 2000 operating system and our modified StarWebService 2.0.2. [4]

First we vary the request frequency from 20 requests per second to 100 requests per second to measure the response time on the client side. The result shows that the response time increase as the request frequency increase and the response time of RRR algorithm is shorter than the active replication algorithm. Then we vary the number of the member from 2 to 8 to measure the response time on the client side. The result shows that the response time of the active replication algorithm increases rapidly as the member number increases and the response time of the RRR algorithm hardly increases as the member number increase. Finally we measure the response time of the RRR algorithm with failure occurring. The result shows that the response time hardly change as the number of the failure member changes.

4 Conclusion and Future Work

In this paper, we presented a new algorithm named RRR. In the algorithm, the response is just returned by the fastest member. So it has shorter response time than the active replication algorithm. We prove this by experiments.

The algorithm achieved speed at the expense of communication overhead. Future work is to optimize algorithm to reduce the communication overhead.

Acknowledgments. This work is supported by The Telecommunication Engineering Institute, Air Force Engineering University Doctor Starting Foundation.

References

1. Flavin, C.: Understanding fault-tolerant distributed systems. Comm. of ACM. 2 (1991) 57-58
2. Schneider, F.: chapter 7: Replication Management using the StateMachine Approach. In: Addison, W. (ed.): Distributed Systems. 2nd edn (1993) 169–197
3. Budhiraja, N., Marzullo, K., Schneider, F., Toueg, S.: chapter 8: The Primary Backup Approach. In: Addison, W. (ed.): Distributed Systems. 2nd edn (1993) 199–216
4. StarWebService 2.0.2. Available online at http://www.starmiddleware.net/ws (2004)