International Journal of Performability Engineering, Vol. 7, No. 2, March 2011, pp.195-197.

Printed in India

Recovering Lagging Replicas in a Fault Tolerant System

HUA CHAI and WENBING ZHAO*
Department of Electrical and Computing Engineering, Cleveland State University
2121 Euclid Ave., Cleveland, OH 44115, USA

(Received on July 14, 2010, revised on November 10, 2010)

Abstract: In this paper, we discuss an often-ignored, but very important issue, *i.e.*, how to recover slow replicas quickly in a fault tolerant system. Despite the fact that the replicas are deployed in identically-equipped computing nodes, under heavy load, some replicas would lag behind due to various reasons. Quickly recovering slow replicas is important because not doing so could result in reduced throughput, high jitters in end-to-end latency, and reduced replication degree.

Keywords: *state-machine replication, fault tolerance, recovery, state transfer.*

1. Introduction

State-machine replication has been of great interest to researchers in the past several decades. Significant progress has been made with regard to the peak performance of normal operation, and prompt and correct recovery from crash failures in such fault tolerant systems [1-4]. However, to ensure sustained optimal operation over long period of time, one cannot ignore the negative impact caused by lagging replicas.

A replica can lag behind other replicas temporarily due to a variety of reasons, such as asynchronous garbage-collection in the Java virtual machine or operating systems, or resource contention with other processes. Once a replica temporarily lags behind, it often leads to the loss of messages due to the overflow of the message input buffer, which aggravates the situation because the replica might be blocked waiting for the retransmission of the missing messages, making the replica lag further behind other replicas. For group-communication-based fault tolerant systems, such lagging replicas would eventually be timed out and removed from the membership, thus reducing the replication degree. For consensus-based fault tolerant systems, such replicas could not participate with the ongoing consensus process, which in effect temporarily reduces the replication degree as well.

In this paper, we focus on a fault tolerant system based on the Paxos algorithm [1] (referred to as Paxos-based fault tolerant system, or PFT in short), due to the efficiency and provable correctness of the Paxos algorithm [1]. The contributions of this paper are two folds: (1) highlight the issue of lagging replicas, and (2) provide our solution on quickly recovering lagging replicas to ensure sustained optimal performance.

2. Investigation of the Lagging Replica Issue

PFT is written entirely in Java and IP multicast is used to enable multicast between the replicas, and between the client and the replicas [5]. We are not aware of any existing Paxos-based fault tolerant systems that have tackled the lagging replicas issue. Hence, in the initial implementation, we adapted

*Corresponding author's email: wenbing@ieee.org

the multicast-based approach introduced in [3]. In this mechanism, each replica periodically multicasts to other replicas its status regarding recent message ordering activities (e.g., message with sequence number 10 has been committed). Upon receiving such a status-update message, a replica compares the status with its own. If it determines that the sending replica has missed a message, it will initiate a retransmission via a point-to-point message. To enable garbage collection of logged messages, each replica periodically takes a checkpoint of its state and shares the checkpointing information with other replicas. When it is sure that a quorum of replicas have advanced to the same state, a replica then garbage-collects all logged messages that have led to the checkpoint. Unfortunately, this may happen when another replica still needs one or more of the discarded messages to proceed. To bring such a lagging replica up to date, a replica in the fast quorum (typically the primary) transfers its state to the lagging replica.

It may appear that the message retransmission mechanism (to handle typical case) and the state transfer mechanism (to handle the worst case) would be sufficient to recover a lagging replica quickly. Unfortunately, during heavy load (in the presence of multiple concurrent clients), the mechanisms are not very effective, as shown in Figure 1(a). After extensive profiling, we found out that the lagging replica often could not receive the retransmitted messages or the transferred state promptly, probably due to the extensive use of multicast messages.

3. Engineering the Fault Tolerant System for Fast Recovery

Our approach (referred to as on-demand approach) relies on combining on-demand message retransmission, a lightweight state transfer, and in the worst case, a state transfer to help a lagging replica recover quickly. Each replica maintains a state log, keeping track of the status of the ordering process for each application request. When it is known that the lagging replica has missed some ordering messages, but the consensus has completed on the relevant messages, a lightweight state transfer is carried out, instead of retransmitting the missing messages because the lagging replica could bypass the ordering process, hence, reducing the recovery time. Furthermore, the use of on-demand retransmission minimizes the use of multicast messages, hence, avoiding aggravating the buffer overflow issue.

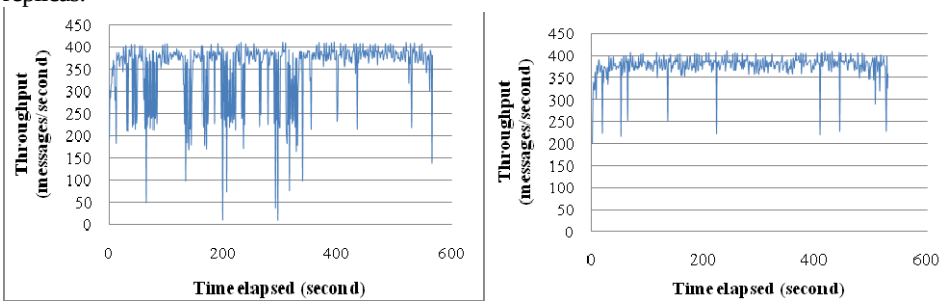
A replica contacts the primary when it experiences a message loss. A lost message can be a control message used to achieve consensus among replicas on the total ordering, or an application request from a client. Upon receiving an ordering request (i.e., Accept or Commit Request), a replica determines if it has missed some messages by comparing the sequence number of the request with the next expected sequence number. The replica would request a retransmission from the primary if a gap in sequence number is detected. Once the retransmission request is sent to the primary, the replica can continue participating in subsequent ordering processes without waiting for the missing messages.

Upon receiving a retransmission request, the primary first checks if the requested messages have been garbage collected. If that is the case, the only way to recover the lagging replica is to initiate a state transfer. If the messages have not been garbage collected, the primary might retransmit the requested messages, or perform a lightweight state transfer if the requested control messages are for message orderings that have been committed.

4. Performance Evaluation

To compare the proposed on-demand recovery scheme with the existing multicast-based approach, we profile the instantaneous throughput of the system, i.e., the number of messages executed within a

certain time interval. In our experiment, the time interval is set to 500 ms, the application state size is 20MB, the replication degree is 5, and the checkpoint interval is set to 500 messages. In each run, we launch 20 concurrent clients, with each client continuously sending 10,000 requests to the server replicas.



(a) **(b)**

Figure 1: Measurements of the instantaneous throughput vs. time elapsed for the multicast-based recovery scheme **(a)**, and our on-demand recovery scheme **(b).**

The evaluation result shows that with the on-demand recovery (in Figure 1(b)), PFT has a more stable instantaneous throughput than the multicast-based recovery approach (in Figure 1(a)). On average, the number of catch-ups (i.e., message retransmissions or state transfers) happening during each run for the multicast-based approach is about twice as much as that of the on-demand scheme (60 vs. 30). And 25 of them require a state transfer to recover the lagging replica in the multicast-based scheme, whereas no state-transfer is necessary when the on-demand scheme is used.

5. Conclusions

In this paper, we have highlighted the importance of recovering the lagging replicas in a fault tolerant system, and proposed an on-demand recovery scheme that can quickly recover lagging replicas and ensure optimal sustained operation of Paxos-based fault tolerant systems.

Acknowledgement

This work is supported in part by an NSF grant CNS-0821319 and by a CSUSI grant from Cleveland State University.

References

[1] Lamport, L. *Paxos made simple*. ACM SIGACT News (Distributed Computing Column) Dec. 2001; 32(4):18-25.

[2] Burrows, M. *The Chubby lock service for loosely-coupled distributed systems*. Proc. of the 7th USENIX Symposium on Operating Systems Design and Implementation 2006; pp. 335-350.

[3] Castro, M. and B. Liskov. *Practical Byzantine fault tolerance*. Proc. of the 3rd Symposium on Operating Systems Design and Implementation Feb. 1999; pp. 173-186.

[4] Zhao, W. *A game theoretical view of Byzantine fault tolerance design*. International Journal of Performability Engineering Oct. 2007; 3(4):498-500.

[5] Zhao, W., H. Zhang, and H. Chai. *A lightweight fault tolerance framework for web services*. Web Intelligence and Agent Systems: An International Journal 2009; 7(3): 255-268.