MicroBlog

Wenjie Huang, Di Ma

1. Introduction

In this project, we implement a micro blog application hosting on EC2 instances. The application is fully replicated through 5 sites. Each site maintains a posted log. The application use Paxos as the consensus algorithm to achieve log consistency. Currently, only post and read operation is supported.

2. Design philosophy

In this project, we provide two version of implementation: basic and optimized. We will discusse the features of these two implementations, why the optimized one has better performance and why it is correct.

2.1 Basic Implementation – single paxos

The basic implementation holds flowing features: 1. logs in five replicas have exactly the same order. 2. Several posts may compete for the same log position. When this situation happens, only one post wins and the others fail. Figure 1 illustrates the application architecture.

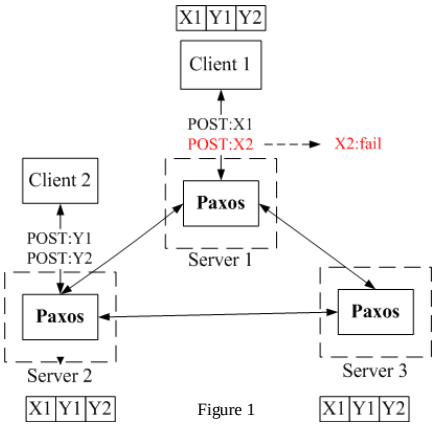

Figure 1

Each server runs one Paxos instance. The Paxos instance runs the Paxos algorithm to reach consensus decision. The decision is global, which means once a decision for log in position i is decided, each replica store the same log in position i. In Figure 1, if msg1 and msg2 compete for the same position, only one post can be accepted and other one will fail.

Prepare and accept phrase: When a message is posted to a server, server will compute the current position and a global unique and monotonic increased ballot number, and broadcast the proposal to all servers. The accept phrase the basically the same with [1].

Learning and recover: When a server receives majority number of accepted messages, it decides the log for current position. In the meantime, it broadcast decided messages [decide:position:post] to all servers. The recover mode will be invoked in 2 conditions: 1. When receives a decided message, compare the current position with the decide position. If current position is smaller than the decide position, there is a “decision hole” between servers. The server with smaller current position will invoke recover mode. 2. When a server is initialized or reboot from failure, it will invoke recover mode.

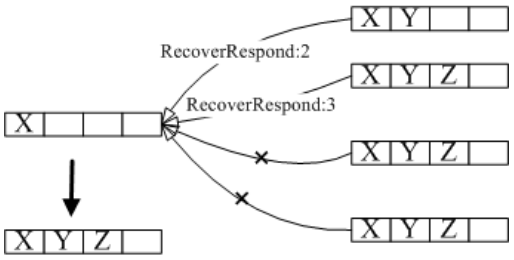

Figure 2

When a server run the recover procedure, it broadcast [recover:position] to all servers. When a server receive [recover : position], it sends back [recover_respond:position], followed by decided messages which position lies between the position sent by recover server and position it knows currently. The recover server collects and counts the recover_respond message. If the number of recover_respond message equal or large than the majority number of sites, it believe that it learns what is the largest decided position and it can catch up with the latest decision from the decided messages flowing the recover_respond message. The reason for checking the number of recover_respond received is to guarantee the server can learn up to the most currently position.

2.2 Optimized Implementation – parallel multi paxos

The optimized implementation holds flowing features: 1. log in five replicas is not necessary in the exactly same order. It only maintain the happen before relation for blog posted in the same server. 2. If no failure, posts from different server can be accepted concurrently and won't compete for the same position.

To achieve these features, each server runs multiple Paxos instances in parallel. Each Paxos instance deal with posts from specific server and won't be interfered with each other. The process of posts is well isolated. Figure 3 shows the application architecture of the optimized implementation.

The number of Paxos instances run in each server is equal to the total number of servers. For the example shown in Figure 3, each server runs three Paxos instances in parallel. We call them Paxos#1, Paxos#2 and Paxos#3. Paxos#1 only

participate in processing posts from Server#1, and same for Paxos#2, etc. In such architecture, each Paxos instance runs basically the same algorithm as basic implementation described above. They are different in ballot number and position computing.

Prepare and accept phrase: each Paxos instance compute a local position and local unique monotonic ballot number instead of a global one. For instance, suppose two clients post X1 and Y1 to Server 1 and Server 2 concurrently. Server 1 use Paxos#1 to prepare the proposal with ballot number [0,1]. Server 2 use Paxos#2 to prepare the proposal with ballot number [0,2]. Server 1 uses Paxos#2 to respond promise to Server 2. Server 2 uses Paxos#1 to respond the promise to Server 1. Although [0,2] > [0,1], X1 will be accepted. Since these two ballot numbers are belong to different Paxos instance, and each Paxos instance works independently. The position is the local position for each Paxos instance. X1 takes position 1 in Paxos#1 in each server, and Y1 takes position 1 in Paxos#2 in each server. For the same reason, the position won't conflict, so two concurrent posts won't compete for the same position.

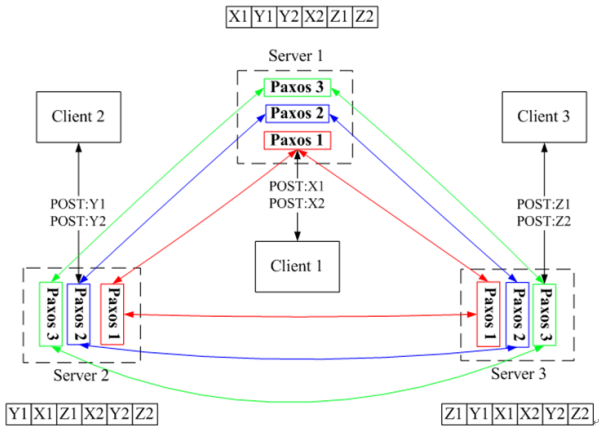

Figure 3

Learning and recover: The learning and recover procedure is same as the basic implementation. When the decide message arrives, server will identify the serverId which the decided post belongs to, and put the decided message into the corresponding Paxos instance. For example, Server 1 first receive decide X1, it use Paxos#1 to decide the value, save it into local log of Paxos#1 and append X1 to the end of global log. After that, Server 1 receive decide Y1 and use Paxos#2 to do the same job for Y1. The global log of Server 1 is X1Y1. For Server 2, it's possible that it first receives Y1, and it uses Paxos#2 to process the message. After that, Server 2 receives X1, and put it in Paxos#1. It's easy to see that Paxos#1 in Server 1 and Server 2 maintain exactly the same log posted by Server1, with same order, and same for Paxos#2. But the global log may differ if Paxos instances receive message in different order. For this case, global log of Server 1 is X1Y1, while Server 2 is Y1X1.

The recover position is local based. If Server 1 needs to recover,Paxos#1 in Server 1 request the latest position from Paxos#1 in other servers, and same for Paxos#2, etc. Each Paxos instance in the recover server request the information from corresponding Paxos instances in other servers.

Correctness: Because every Paxos instances are followed the rule of Paxos, the order of the messages from the same server can be guaranteed, which promised the correctness of the causal relations of these messages. On the other hand, messages from different server do not have any causal relation; the messages from different servers can be in any order, which means that five Paxos instances work paralleled will not cause any false causality. In conclusion, the optimized Paxos is correct and the performance is improved.

Performance: The optimized Paxos has better performance than the basic Paxos. It improves the throughput of the program, and avoids the ballot number conflict problem in the basic Paxos. Because there are several Paxos Instances in the optimized Paxos server, each instance will only communicate with its corresponding Paxos instance. In this situation, five Paxos Instances can process the messages from five servers parallel, which will certainly have higher throughput than the basic one, which has only one Paxos instance. What's more, because the messages from different servers are processed separately and each Paxos instance has its own local ballot number, the message from different server will not compete for one ballot number. As a result, there is no ballot number conflict between messages from different server. Since messages from the same server will not have ballot number conflict, there is no ballot number confliction between optimized Paxos servers.

3. Deployment

The program is running on the micro instances with Ubuntu 13.10 and openJDK 7 in EC2. There are 10 servers running on 10 micro instances, which are deployed on five different regions: Virginia, Organ, California, Ireland, and Singapore. Basic implementation are deployed on five of these instances, the other five instances deploys optimized implementation.

Reference:

[1] T. Chandra, R. Griesemer, and J. Redstone. Paxos made live-an engineering perspective (2006 invited talk). In Proceedings of the 26th ACMSymposium on Principles of Distributed Computing-PODC, volume 7, 2007.
[2] L. Lamport. Paxos made simple. ACM Sigact News, 32(4):18–25, 2001.