DAT520 Exam Report

Comparative analysis of Multi-Paxos, Raft and Compartmentalization

Lejun Chen
University of Stavanger, Norway
l.chen@stud.uis.no

ABSTRACT

This article demonstrates two consensus protocols, Multi-Paxos and Raft. We analyse the similarities and differences between them. The pros and cons are also discussed. At the end of this article, three techniques for improving the performances (throughput and latency) are introduced. They are pipeline, batching and compartmentalization. We also do a comparative analysis between these three techniques.

KEYWORDS

Paxos, Multi-Paxos, Raft, Consensus protocol, Compartmentalization, Pipeline, Batching

1 INTRODUCTION

Consensus algorithm is the critical element for designing a highly reliable and available distributed system to tolerate failures and agreeing for the values. Paxos [1, 2] is probably the most popular consensus protocol. However, it is difficult to understand and implement due to its complicated mechanism. Raft [3] is designed for solving the same problem with the goal of being understandable. In the following sections, the basic Paxos protocol is demonstrated first, then extended to Multi-Paxos. Raft is illustrated in detail afterwards.

The complex mechanisms of consensus algorithms lead to low throughput and high latency. Thus, many techniques are proposed to improve the performances. At the end of the report, we will discuss three of them, pipeline, batching and compartmentalization [4].

2 PAXOS

2.1 System model & underlying assumptions

Paxos operates under the following system model & underlying assumptions:

- Network communication is asynchronous and unreliable. Messages on the network can be lost or delayed. But the arrive ones are not corrupted.
- The servers may crash, stop and restart but when they are running, they always behave correctly.
- There is no malicious node (Byzantine failures) in the system.
- Servers are implemented as deterministic state machines. All start from the same initial state. The next state is completely determined by the current state and the update being executed.
- Client issues a new request only when it receives the response for the last request.

Exam Report (DAT520), UiS,
2021.

2.2 Overview

A high-level structure of Paxos server is illustrated in Figure 1. There are four main parts in a Paxos server:

- *Leader election module*: Contain two sub-modules, failure detector and leader detector. It selects a leader to coordinate the consensus procedure.
- *Consensus module*: Contain three components, proposer, acceptor and learner. They can be run in the same or different machines. In this section's demonstration, we assume each server run all three.
- *Log*: Store the decided values. The aim of consensus algorithm is to have the log identical for each server.
- *State machine*: Execute the decided value and send back response to clients.

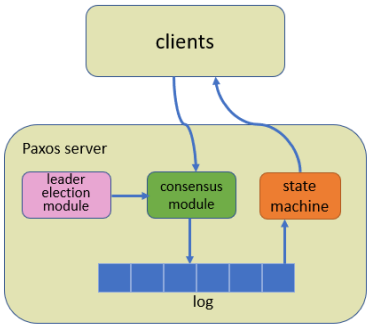

Figure 1: Paxos server architecture.

2.3 Basic paxos

Leader election

In Paxos, leader election module contains failure detector and leader detector. Every server sends heartbeats periodically to other nodes. The failure detector collects these heartbeats to check others is alive or not. This information will be provided to the leader detector. It selects the leader according to certain criterion, e.g., the node ranking, etc.

Consensus

After leader has been elected, it starts the consensus procedure, which could be divided into two phases.

Phase 1. As shown in Figure 2, the proposer selects a unique proposal number(called ballot) and sends Prepare RPCs (Phase 1a message) to all acceptors. Up receiving a Prepare RPC, if the acceptor has not seen a higher ballot, it replies with Promise RPC (Phase 1b message) containing highest ballot it had ever accepted.

If the proposer receives a majority of Promise RPCs, it will start Phase 2, Otherwise, retry with a higher ballot.

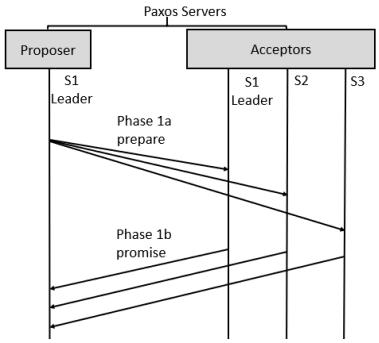

Figure 2: Message sequence of Phase 1 of Paxos.

Phase 2. As shown in Figure 3, proposer picks proposal value and broadcasts Accept RPCs (Phase 2a message) to all acceptors. Upon receiving Accept RPCs, if the acceptor has not seen a higher ballot, it will send Learn RPCs (Phase 2b message) to all learners containing proposed value. Upon receiving a majority of Learn RPCs, the learner will consider the proposed value as a decided value. The state machine replica executes this decided value and sends back response to the client.

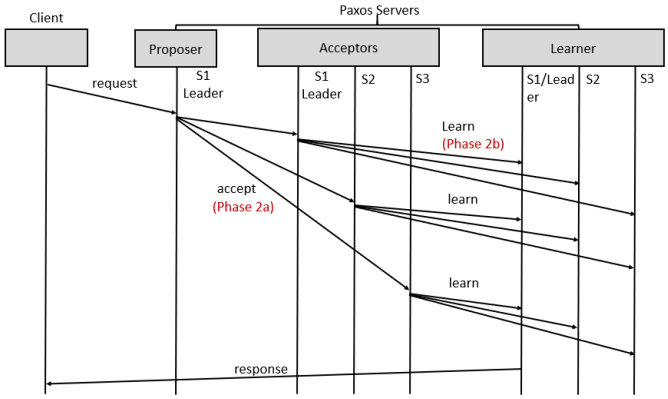

Figure 3: Message sequence of Phase 2 of Paxos.

2.4 Multi-Paxos

Single-Paxos chooses only one single request at a time, agrees on value one by one, as shown in Figure 4. Multi-Paxos chooses many values at a time, and decides them in parallel, as shown in Figure 5. Multi-Paxos is significantly efficient than Single-Paxos especially in high latency systems.

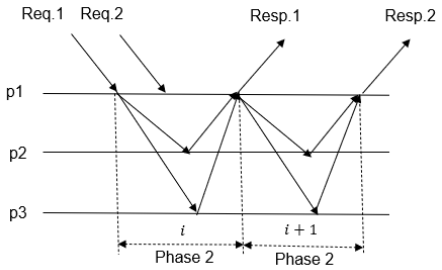

Figure 4: An example of Single-Paxos's Phase 2. Slot i+1 consensus instance starts after the slot i has finished.

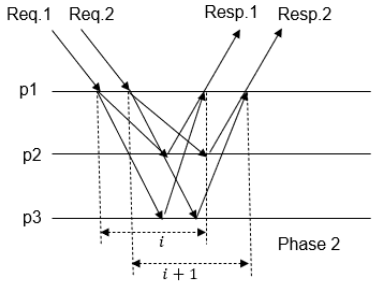

Figure 5: An example of Multi-Paxos's Phase 2. Slot i and slot i+1 consensus instances are agreed in parallel. The Response 1&2 are sent back to clients.

3 RAFT

3.1 System model & underlying assumptions

Paxos Raft has similar system model and underlying assumptions as Multi-Paxos, see section 2.1. But it has some special assumptions because of its own characteristics:

- Broadcast Time is much smaller than election Timeout. If this requirement is not meet, it is likely to happen that a new vote starts before other nodes receive the voting requests for the last vote. This will result Raft cannot elect and maintain a steady leader.
- Leader never overwrites or deletes its log entries, only allows to append.

3.2 Overview

A high-level structure for a Raft server is shown in Figure 6. The main difference from paxos server is that the leader election module is integrated into the consensus module.

Raft divides time into terms, see Figure 7. Each term begins with a leader election (Phase 1). Log replication (Phase 2) proceeds. The Phase 2 repeats until the leader becomes crash or unreachable. Then a new term starts. If no leader is elected in the current term, term number will be increased, a new election will be started.

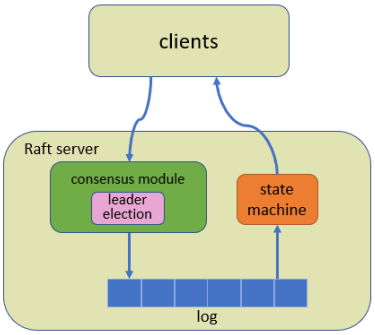

Figure 6: Raft server architecture.

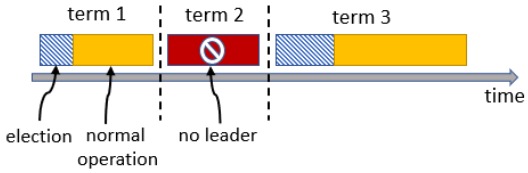

Figure 7: Raft's term.

3.3 Phase 1: Leader election

Election mechanism

Raft has a different election mechanism with Multi-Paxos. In Raft, the server has three states, follower, candidate and leader, see Figure 8.

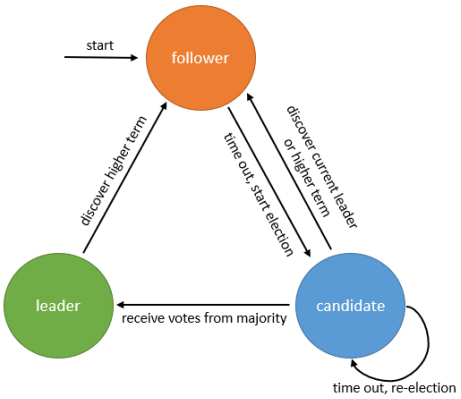

Figure 8: Raft server switches among three states: follower, candidate and leader. It converts to another state when a certain criterion meets.

At beginning, each server starts as a follower. Each follower has a random election timeout. During election timeout, if the nodes receive heartbeats, it will reset the timeout clock, otherwise, it will time out and convert to candidate state. In raft, the two types of RPCs (RequestVote & AppendEntries) are all considered as heartbeats.

It worth pointing out that the random timeout could prevent all followers from turning into candidates at the same time.

Figure 9 shows the message sequence of Phase 1. Candidate votes for itself, increases the term number, issues RequestVote RPCs which include the information of log entries and sends them to other servers for voting.

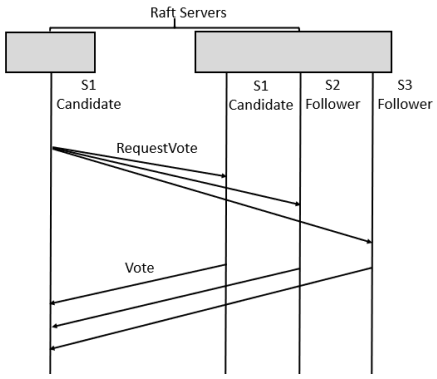

Figure 9: Phase 1 of Raft's consensus procedure. A example of three servers: S1 as a candidate, S2 and S3 are followers.

Election outcomes

There are two election outcomes, referring again to Figure 8:

(1) *Receive majority votes.*
Candidate receives a majority of votes if its log is more updated. It becomes leader and start Phase 2.

(2) *Election timeout.*
- The current candidate keeps being a candidate, hold a new election.
- Another node time out earlier than the current candidate, converting to be a candidate. It increases term number and starts a new election. The RequestVote RPCs will be sent by this node. The current candidate, receiving this valid heartbeat, steps down to a follower.

3.4 Phase 2: Log replication

Figure 10 shows the message sequence of Phase 2. After the leader has been elected, it accepts request from the client and store it in its log. Log entry is a tuple including index, term and request. Leader propagates log entries to the other follower by issuing AppendEntries RPCs.

A follower rejects the AppendEntries RPC if it has seen a higher term. Otherwise, it confirms and stores that log entries. Raft always assumes the leader has the correct and most updated log. If the followers' log has conflict with leader's, it will erase its log entries in conflict, synchronize its log with the leader's.

Once the leader receives a majority of confirmations, that log entry is considered as committed. Leader will inform other nodes this information. Replicas only execute the committed log entries.

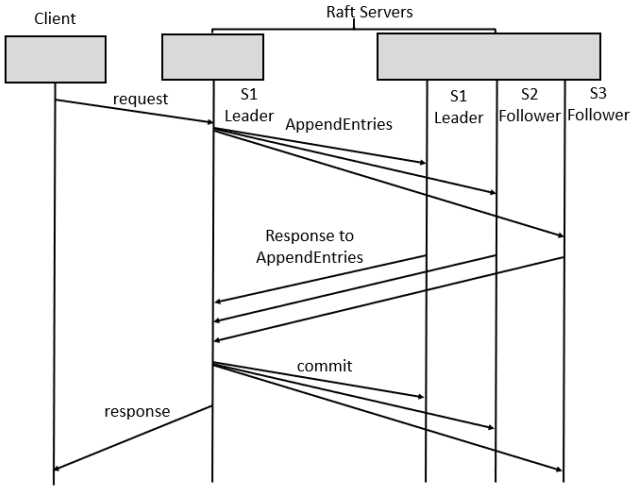

Figure 10: Phase 2 of Raft's consensus procedure. A example of three servers: S1 as a leader, S2 and S3 are followers.

4 SIMILARITIES & DIFFERENCES BETWEEN MULTI-PAXOS AND RAFT

4.1 Similarities

Raft and Multi-Paxos have many in common:

- Operate under similar system model.
- Have similar architectures: leader election, consensus module, log and state machine.
- The log index of Raft is similar to the instance ID (slot ID) of Multi-Paxos.
- The term in Raft is similar to proposal number(ballot) in Multi-Paxos, which indicates the changing of leadership.
- Both two protocols tolerate n failures with 2n+1 servers.

4.2 Differences

Table 1 summarizes the Differences between Multi-Paxos and Raft. The differences in components, election module, election criterion, server states and consensus phase are intuitive, we won't demonstrate here anymore. The differences in leader, heartbeat, log continuity are discussed as below.

Leader. Raft assumes at most one leader at any time, and proposals can only be sent by a leader (strong leader). Otherwise, the correctness is affected. Multi-Paxos also elects leaders, but it is only for efficiency.

Heartbeat. In Multi-Paxos, heartbeat is a simple message only for the purpose of showing the node is alive or not. In Raft, RequestVote RPCs and AppendEntries RPCs not only serves as heartbeats but also for voting and log replication.

Log continuity. When the followers received AppendEntries RPCs, it will check the log continuity before confirming it. If it detects the discontinuity, the AppendEntries RPCs will be rejected. Then the leader will send another AppendEntries RPCs including more log entries, until the log continuity is detected by followers. Multi-Paxos skips this check.

5 PROS & CONS OF MULTI-PAXOS AND RAFT

Table 2 summarizes merits and demerits of Multi-Paxos comparing with Raft. Multi-Paxos is better in popularity, availability and robustness, while Raft performs better in understandability, efficiency and simplicity.

Table 2: Summary of Pros and cons of Multi-Paxos &Raft

	Multi-Paxos	Raft
Popularity	×	
Understandability		×
Efficiency		×
Availability	×	
Simplicity		×
Robustness	×	

* "×" means a better performance for corresponding row.

Popularity. During two decades' development, Paxos is widely used and has many variants. These Paxos variants range from reducing latency, increasing throughput, balancing replica load, to tolerating Byzantine faults etc. Raft was first proposed in 2014. Although it is becoming more and more popular, Paxos is still the dominant consensus protocol.

Understandability. Paxos was difficult to implement for its complicated mechanism. Many literature were published trying to illustrate it and make it more understandable. The motivation for proposing Raft is for understandability. It has less components, message types and simpler consensus procedure, thus a better understandability.

Efficiency. We discuss efficiency in the following two aspects:

- *Network load*: During the consensus procedure, Raft use fewer messages than Multi-Paxos. During leader election, in Multi-Paxos, heartbeats sent by every server periodically, significantly increase the network load. In Raft, the two RPCs (RequestVote & AppendEntries) serves as heartbeats, no need to design messages specifically for heartbeats.
- *Latency*: Multi-Paxos requires at least two network trips to reach an agreement while Raft just need one single round-trip.

Availability. When there is no leader or multiple leaders in both two system, Multi-Paxos is still available while Raft is not. Moreover, the follower must wait for the lease of the old leader to expire before launching an election. But in Multi-Paxos, the leader election is run in parallel with consensus procedure, responding faster than Raft when the leader is not reachable. Thus, Multi-Paxos has better availability than Raft.

Simplicity. Raft's leader election is significantly lightweight comparing with Multi-Paxos. The Multi-Paxos mechanism is obviously much more complicated than Raft. It has three components; more messages need to be communicated. Thus, Raft is simpler than Multi-Paxos.

Table 1: Differences between Multi-Paxos and Raft

	Multi-Paxos	Raft
Leader	Weak leader	Strong leader
Components	Proposer,acceptor, learner	None of these components
Election module	Separated	Incorporated
Heartbeat	A simple message indicating the nodes alive or not	RequestVote RPCs & AppendEntries RPCs
Election criterion	Can be any criterion	The most-updated log
Server states	Leader, non-leader	Leader, candidate, follower
Log replication	Check continuity	Do not check
Consensus phase	Phase 1 & Phase 2	Phase 2

Robustness. Raft assumes that the system has at most one leader at any time, and proposals can only be sent by a leader (strong leader). Multi-Paxos also elects leaders, but this is only for efficiency. Multiple leaders or no leader do not affect the correctness. Multi-Paxos does not limit that only the leader (weak leader) can propose proposals. Thus, Multi-Paxos is more rubust.

6 POTENTIAL USE

The consensus protocols had been used in many fields, e.g., state machine replication, state estimation, load balancing, control of UAVs (and multiple robots/agents in general), smart power grids, clock synchronization, opinion formation, cloud computing, PageRank, etc. One of the evident applications can be seen in the blockchain [5].

7 TECHNIQUES FOR IMPROVING PERFORMANCE

7.1 Background

The existing consensus protocols involve relatively complex mechanisms and a large number of communication messages. All these lead to poor performance, like low throughput and high latency. The unreliable network makes things even worse. Thus, many techniques are proposed to improve the performances. In this section, we will discuss three of them, pipeline, batching and Compartmentalization.

For the sake of better demonstration, we use the Multi-Paxos as an example to illustrate them. Specifically, we discuss their application in the Phase 2 since the normal operation in consensus procedure just involves Phase 2. The Phase 1 can be optimized to skip when the leader is stable. However, these three techniques can be easily adapted to other consensus protocols.

7.2 System Model

All three techniques operate in same system model as Multi-Paxos since we now apply them in Multi-Paxos protocol. We do not repeat it here, please refer to section 2.1.

7.3 Pipeline

In section 2.4, we discussed how to extend the Basic-Paxos to Multi-Paxos. The protocol choose many values at a time, and decide them in parallel. This technique is called pipeline. This optimization is effective in reducing the latency, as the leader might have to wait a long time to receive the Phase 2b messages. One important thing for pipeline is how to choose the optimal bound for the number of parallel instances. The resource will not be utilized sufficient with a too low number. But for a too high number, the system will be overloaded resulting in a performance degradation. It usually requires careful analysis based on experiments.

7.4 Batching

As showed in Figure 11, the leader waits for a certain time duration, collects "enough" requests, packs as a single consensus instance(batch). After the batch is agreed, it is unpacked, executed and send back to clients.

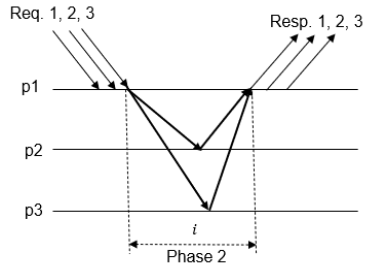

Figure 11: A example of batching applied to Phase 2 of Multi-Paxos. Request 1, 2 & 3 forms a batch. A thick line indicates message containing a batch.

The benefit of batching is many requests share one consensus instance. Thus, the consensus cost and overhead are greatly reduced for each request, a higher throughput is achieved. Additionally, batching decreases storage cost. We need n log entries for n requests without batching, but now we just need one, given n the batch size.

The main drawback is it may cause possible delay for these early arrived requests when the leader is waiting for "enough" requests. Moreover, longer time will be required for transferring larger consensus instance. Obviously, low load system does not suit batching as it may take a long time to wait for "enough" requests.

A larger batch size probably increases the throughput but makes the latency higher. Thus, the size of batch and the time point for

packing must be carefully tuned. Some researches [6,7] had already been done about this problem.

7.5 Compartmentalization

Basic concepts

For the simplicity and generality, we adapt phase 2 procedure as shown in Figure 12 for the sake of better demonstration. We assume MultiPaxos deploys 2f + 1 machines with each machine has all three roles of a MultiPaxos proposer, acceptor, and replica.

In Multi-Paxos, the components are responsible for handle multiply tasks and process different types of message. The main idea of compartmentalization is to decouple these tasks then scale the newly added components, thus balance the workload of each node, eliminate the bottle neck and increase the throughput.

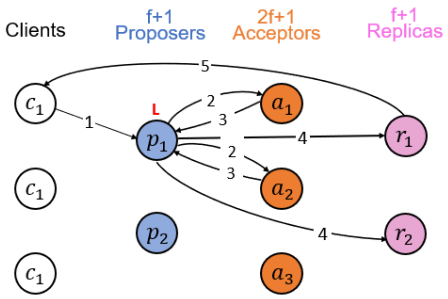

Figure 12: Phase 2 of Multi-Paxos. The letter "L" in red colour represents the leader. Message 1 is command from clients, Message 2 is phase 2a message (Accept RPC) , Message 3 is phase 2b message (Learn RPC) , Message 4 is decided value, Message 5 is response to client.

Step 1 Decouple

(1) *Leader.*
In Phase 2, the leader in Multi-Paxos is responsible for receiving the commands from clients, sequencing them, then broadcasting to acceptors. We could separate these tasks by introducing f+1 proxy leaders, as shown in Figure 13. Now the leader is just responsible for receiving and sequencing commands. The proxy leaders broadcast the Accept RPCs to acceptors and collect a quorum of (f + 1) responses to Accept RPCs.
Thus, the leader just processes 2 messages. Without decoupling, it needs to process 3f+4 messages per command. It makes the leader much less likely a bottleneck.

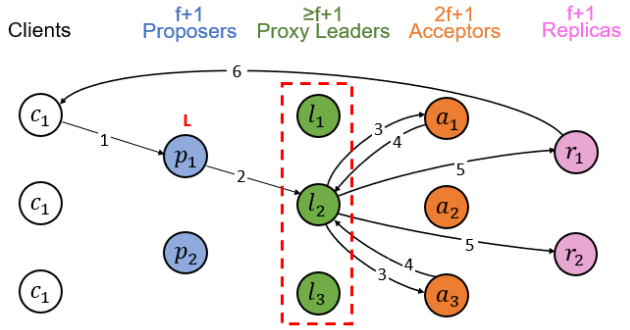

Figure 13: Introduce Proxy leaders (in dash red rectangle) to decouple Proposer.

(2) *Acceptors.*
The 2f+1 acceptors also could be a bottleneck. To prevent it, we add more groups of acceptors, each group has 2f+1 acceptors, as shown in Figure 14. The added acceptors groups also could perform response to Phase 2a messages. Thus, work load could be reduce to 1/n, given n acceptor groups.

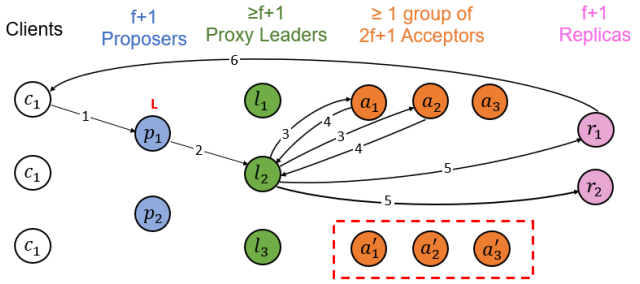

Figure 14: Introduce more acceptors groups (in dash red rectangle) to decrease the load on each acceptor groups) to decouple Proposer.

(3) *Replicas.*
Every replica could send back the response since all state machine replicas have same state. Thus, given n replicas, every replica only need to send back 1/n responses of all commands. However, we could reduce this workload even more by increasing the number of replicas, as shown in Figure 15. This helps to eliminate the Replicas as bottle neck.

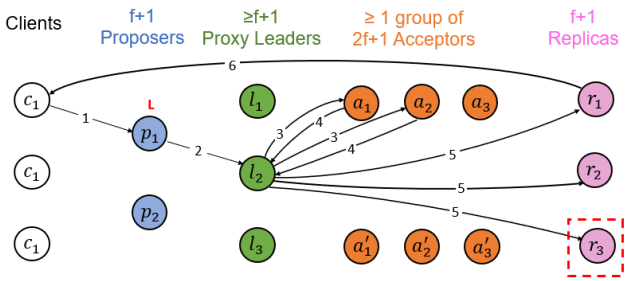

Figure 15: Introduce more replica (in dash red rectangle) to decrease the load on each replica.

(4) *Read paths and write paths.*
We call commands that modify the state of the state machine writes and commands that don't modify the state of the state machine reads [4]. Before decoupling, all writes and reads message are coordinated by leader. However, if the clients communicate the reads messages directly with accepotors and replicas, it won't cause the divergency of the replica's state. In another word, it is not necessary for leader to processing the reads message. Thus, we could decouple read path and write path as Figure 16 shows. This decreases the number of messages that leader need to process, help to prevent the leader to be the bottle neck.

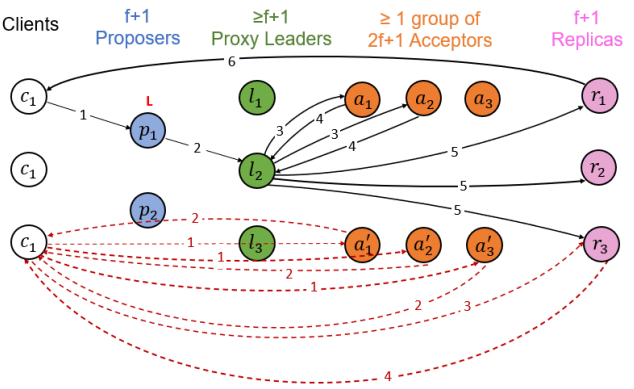

Figure 16: Separate read paths from write paths.Black solid lines are write paths, red dash lines are read paths. The read messages are directly communicate with acceptors and replicas, no need to follow the routes for consensus.

(5) *Batching.*
The batching technique also could be applied to compartmentalized Multi-Paxos, as illustrated in Figure 17. The throughput increases further with batching. Two agents(batcher & unbatcher), are introduced to decouple Multi-Paxos with Batching.

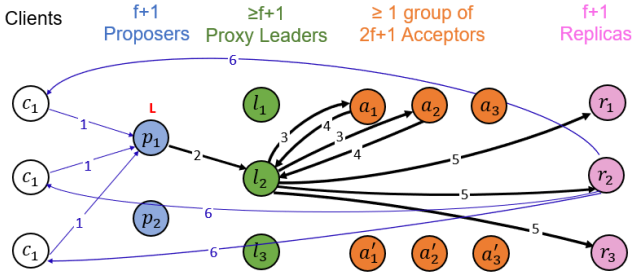

Figure 17: Batching technique applied to compartmentalized Multi-Paxos. A thick line indicates the message containing a batch. A thin line indicates a single request or a single response.

- Batchers.
Refer again to Figure 17, the leader's proposer p_1 is responsible for forming batches and sequencing batches. These two responsibilities can be decoupled by introducing a set of at least f + 1 batchers, shown in Figure 18. The batchers now take responsibilities for receiving requests randomly from the clients, forming batches, then sending them to p_1. p_1 now is just responsible for sequencing the batches and sending Phase 2a messages. Without batchers, p_1 receives n requests per batch. Now it just receives one, resulting in greatly reduction in the load of leader.

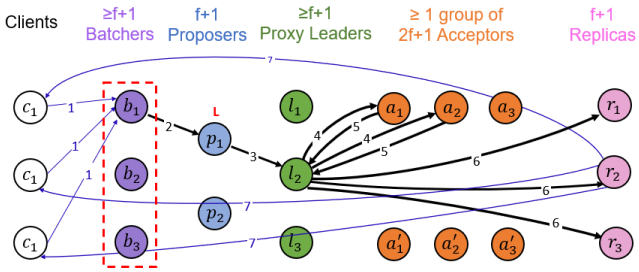

Figure 18: Introduce Batchers (in dash red rectangle) to decouple Multi-Paxos with batching.

- Unbatchers.
Refer again to Figure 17, the replicas are responsible for unpacking batches, executing requests in a batch , and sending the responses to the clients. The unbatchers are introduced to take the responsibility of sending back these responses as demonstrated in Figure 19. Thus it reduces the workload of replicas, eliminating replicas as bottle neck.

Step 2 Scale

Scaling in compartmentalization means how to choose the number for these newly added components (the components in red dashed

Table 3: Pros and Cons comparing Pipeline, Batching, Compartmentalization

	Pipeline	Batching	Compartmentalization
Pros	• Easy to implement. • Increase throughput. • Decrease latency. • Has a significant gain in in systems with moderate to high network latency [6].	• Easy to implement. • Increase throughput. • Save storage. • Provides significant gains both in high and low latency networks [7].	• Use more machines. • Increase throughput. • Decrease latency [4]. • Can scale.
Cons	• Do not gain much in low latency system. • Can not help with eliminating the bottlenecks.	• Increase response time. • Reduces the CPU time available for other services. • Can not help with eliminating the bottlenecks. [7].	• Use more machines. • Complicated to implement and scale.

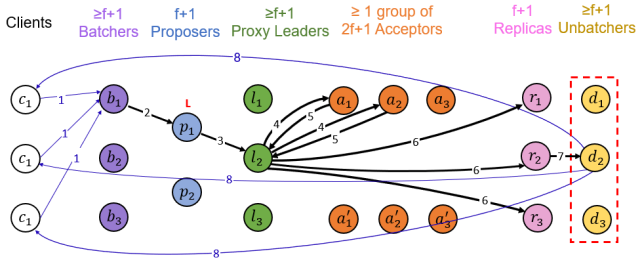

Figure 19: Introduce Unbatchers (in dash red rectangle) to decouple Multi-Paxos with batching.

rectangles in Figure 13-19) to meet the requirements of the systems, if possible, with least resources. There are six decoupling methods discussed above. It is possible that the performance's requirements have already been met although we just choose some of them. All these depends on the careful analysis for specific scenario.

Literature [4] shows us the experimental methods for scaling. It gradually increased the number of proxy leaders, replicas, batch size, unbatchers and recorded the experiment results (throughput latency) for every scaling.

Literature [4] also indicates that throughput is scaled almost linearly with the number of machines. When the number of machines has been scaled up to 6.66×. The throughput has a 6× increase without batching and a 4× increase with batching.

7.6 Pros & Cons of three techniques

These three techniques could be use alone, or in combination of any of them. However, they have constraints because of their different working mechanism. The pros and cons are listed in Table 3.

From Table 3, we could see all of them increase the throughput, however batching will increase the response time as well. Pipeline is not suitable for low latency system. Since the instances are agreed fairly fast, there will be little differences for executing them one after on or in parallel.

Batching save storage, since with batching, the log just need to store one instead of n(batch size) for the decided value. Howerver, the moment the batch is decided, the server need to execute the n requests almost at the same time. It will take up many CPU capacity, resulting in response slowly to other requests from clients.

Pipeline and batching do not require additional resources, while compartmentalization does. The requirement for more machines is a demerit for compartmentalization. However, this lead to eliminate the possible bottle necks by scaling, adding more machines to system. Thus, needing more machines also can be considered as a merit.

8 CONCLUSION

To choose which consensus protocol and which techniques for improving the performance is not an easy task. There is always a tradeoff. And different techniques have different application scenarios. Before we choose the schemes, we need to fully analyse the characteristics and requirements of the system and the resources available. During the evaluation period, the tuning experiments are need to be carefully designed to achieve the possible best optimization.

REFERENCES

[1] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.
[2] Robbert van Renesse and Deniz Altinbuken. Paxos made moderately complex. *ACM Comput. Surv.*, 47(3):42:1–42:36, 2015.
[3] Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In Garth Gibson and Nickolai Zeldovich, editors, *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*, pages 305–319. USENIX Association, 2014.
[4] Michael Whittaker, Ailidani Ailijiang, Aleksey Charapko, Murat Demirbas, Neil Giridharan, Joseph M. Hellerstein, Heidi Howard, Ion Stoica, and Adriana Szekeres. Scaling replicated state machines with compartmentalization [technical report]. *CoRR*, abs/2012.15762, 2020.
[5] Sivleen Kaur, Sheetal Chaturvedi, Aabha Sharma, and Jayaprakash Kar. A research survey on applications of consensus protocols in blockchain. *Secur. Commun. Networks*, 2021:6693731:1–6693731:22, 2021.
[6] Nuno Santos and André Schiper. Tuning paxos for high-throughput with batching and pipelining. In Luciano Bononi, Ajoy Kumar Datta, Stéphane Devismes, and Archan Misra, editors, *Distributed Computing and Networking - 13th International Conference, ICDCN 2012, Hong Kong, China, January 3-6, 2012. Proceedings*, volume 7129 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2012.
[7] Nuno Santos and André Schiper. Optimizing paxos with batching and pipelining. *Theor. Comput. Sci.*, 496:170–183, 2013.