I. INTRODUCTION

Consensus algorithms is one of the central topics that has been extensively researched in the history of distributed computing. Achieving a certain level of consistency for multiple service nodes in a system is an unavoidable problem in the design of all distributed systems. In 1999, a computer scientist Eric Brewer has published an article (Fox & Brewer,1999) where CAP Theorem was introduced, it is considered as the fundamental theorem to understand distributed systems. CAP Theorem is the acronym for consistent, available, and partition tolerance and it states that at most two of these features can be satisfied in a distributed system. Consistent emphasis on passing correct data between all system nodes, availability emphasis on all client requests will receive a response, and partition tolerance emphasis on the normal operation of the server no matter if there is failure. Distributed systems we use today is often based on asynchronous message passing across the internet, where processes could be delayed, disordered, or lost during passing. These problems can be solved by consensus algorithms because it ensures the consistency of the system by obtaining agreement among many service nodes on a proposal. This paper will introduce two major consensus algorithms Paxos, and Raft algorithms, and compare the differences from the perspective of efficiency, structure, and safety.

II. Paxos

Paxos is one of the earliest consensus algorithms, it was first proposed by Leslie Lamport in 1990. In the article, he wrote a story about the Paxos island as a metaphor for the consensus issue in the asynchronized systems. There are serval roles in the Paxos island, and they are only responsible for their job. Everything on the island is decided by some special people. Legislators will propose laws that apply to all citizens in the chamber, and they will communicate to all other legislators by passing messages. And they will record every message they receive on their ledger. Then they will vote on the

purpose, the law will apply once more than half of the legislators agree to the most recent purpose recorded on their ledger. (Lamport, 2019) This process would be easy if all legislators and messengers are reliable, but they sometimes leave the chamber for a coffee break. Therefore, we can not guarantee all legislators are voting on the same proposal since we do not know if they received the right message. Paxos is the algorithm that ensures the voting process works correctly and the laws passed do not contradict each other. This metaphor is difficult to understand, so Lamport published another version of this article to explain Paxos algorithm in detail. He classifies the roles in the Paxos island as proposer, acceptor, and learner. And assumed that agents can communicate with one another by sending messages, in which "agents operate at arbitrary speed, may fail by stopping, and may restart. Since all agents may fail after a value is chosen and then restart, a solution is impossible unless some information can be remembered by an agent that has failed and restarted. And messages can take arbitrarily long to be delivered, can be duplicated, and can be lost but they are not corrupted" (Lamport,2001). Proposers create a message-passing tunnel between clients and acceptors, they deliver proposals including the proposal ID and the proposal value to each acceptor. The acceptor is the decision-maker, they accept or deny the proposal and send back their decision to the proposer. And the learner learns from the acceptor. If the majority of acceptors have accepted a proposal, then the final result is obtained. The Paxos process can be divided into prepare phase and accept phase. In the prepare phase, "(a) A proposer selects a proposal number N and sends a prepare request with number N to a majority of acceptors. (b) If an acceptor receives a prepared request with a number N greater than that of any prepare request to which it has already responded, then it responds to the request with a promise not to accept any more proposals numbered less than N and with the highest numbered proposal (if any) that is has accepted. In accept phase (a) If the proposer receives a response to its prepare requests from a majority of acceptors, then it sends an accept request to each of those

acceptors for a proposal numbered N with a value V, where V is the value of the highest-numbered proposal among the responses. (b) If an acceptor receives an accept request for a proposal numbered N, it accepts the proposal unless it has already responded to a prepare request having a number greater than N" (Lamport,2001). The first phase attempt to submit a proposal while checking if there is a value selected and block all the old proposal to reduce competition. The second phase performs the real submission. As we see in these two phases, proposal ID is a critical role to ensure the consistency of this algorithm. Because if there are multiple proposers proposing to different acceptors at the same time and none of the proposals were accepted by more than half of the acceptors, proposal ID needs to be implemented to reach consensus. This would be easier to understand if this process is illustrated in a real-life situation. We will explain these two phases by simulating the process of buying a company co-owned by three owners.

I. Case 1: One buyer

The offer was made that they would buy the company for 1 dollar and sent an email to notify each owner. Since no other buyer is competing for this company, this proposal will pass without any obstruction. Owner A, B, and C will respond to the buyer with an accepted message and once the buyer has received two responses contract will be signed because the majority of the owner has accepted the offer. In this case, the buyer is the proposer, the owners are the acceptor, and the price of the company is the proposal ID.

II. Case 2: multiple buyers

Assume there are two buyers both interested in buying this company, buyer A started by offering with 1 dollar, and buyer B started by offering with 2 dollars. Under normal circumstances, messages are delivered correctly without losing. But in the reality, messages could be lost, or

acceptors could be interrupted. Assume buyer A send out their offer first but only owner A and B have received the message, and buyer B send out their offer right after buyer A, but only owner B and C received it. First, buyer A will receive two accept messages from owner A and B since this is the first offer they have seen. Then owner B will get the updated offer from buyer B and make a promise that they will not take buyerA's offer since it has a lower price. When buyer A come back with the contract, they will get rejected by owner B. Since buyer A does not have the majority, no contract is made at this point. And when buyer B comes back with the contract because buyer B and C have previously agreed on their offer, the majority is formed. Therefore, buyer B will sign the contract。

These two cases have proved that the consistency in the Paxos algorithm, the learner will receive the same value eventually. However, live lock problem exists in the Paxos that is if buyer A and B constantly send offers to company owners with a higher price. Owners will keep accepting their offer so no prosper submit successfully. One of the solutions is to elect a leader among proposers, and all other proposers will pass their proposal to the leader, then passes to the acceptor. The leader has selected by-election, if the leader works correctly, it will re-lease its time period after its timeout. Or it will run another election to elect a new leader. This idea was introduced in 2004 by Lamport as Fast-Paxos (Lamport, 2006). The Basic-Paxos was considered as low efficiency because multiple proposers exist at the same time, conflicting proposal occurs frequently, and passing a proposal requires two prepare phase and accept phase. In order to fix this live lock problem and increase the efficiency of this algorithm, a leader election was implemented. The idea is to elect a proposer with the biggest proposal ID as the leader and implement a heartbeat connection to connect all proposers. Proposers will continuously send out their proposal ID and check their responses. If they did not receive

any ID that is higher than its own ID, then it can switch the role to be the leader. All proposers will pass their proposal to the leader and the leader will pass the message to the acceptor. fast Paxos allow multiple leaders to propose concurrently. Ideally, prepare phase can be skipped since only one proposer is passing the message to the acceptor, conflict will not happen without competition. Thus, the problem of live lock could be solved. Therefore, we obtain a consensus algorithm that guarantees both security and aliveness.

III. Raft

The raft algorithm was proposed in 2013 by Diego Ongaro and John Ousterhout，” Raft is a consensus algorithm for managing a replicated log. It produces a result equivalent to (multi-)Paxos, and it is as efficient as Paxos, this makes Raft more understandable than Paxos” (Ongaro & Ousterhour, 2013). Raft defined three roles as leader, candidate, and candidate. The leader is responsible for communicating with clients on behalf of the cluster and only one leader can exist in a cluster. The follower responds to the leader’s request, they can suggest themselves be the new leader if needed. And the candidate is a temporary role that participates in the leader elections. Raft is considered easier to understand than Paxos because in Diego Ongaro and John Ousterhour’s study (Ongaro & Ousterhour, 2013). They decompose the consensus problem into three independent subproblems, which are leader election, log replication, and safety. In the leader election stage, all the servers start as follower and they will be assigned to a random timeout to avoid proposed elections at the same time. Once there is no leader, follower can not keep the heartbeat with the leader, then the server with the shortest timeout will switch its state to a candidate and then ask others to vote, servers will always vote for candidates has a larger term. The server that has the majority of the votes will switch to the leader and constantly send out heartbeats to block other followers from starting a new election. In the log

replication stage, the leader will save the client message it has received into the local log. Then it will send out the message to follower and ask them to replicate this message. Once the message is replicated by the majority of the servers, the leader will include this message into its state machine. Then the leader will send back a success message to the client. And last, the leader will send a check message to all the followers including the highest index it has committed. Then the followers will be forced to replicate leader's log entry into its local state machine. This guarantees consistency between each server's node. The safety stage adds restrictions, on the election, committing entries from previous terms, and safety argument.

IV. Comparison between Paxos and Raft

In a distributed system, some servers may crash or become unreliable because of carious unexpected possibilities. Both Paxos and Raft are designed to achieve consensus, but they achieve it in different ways. Paxos is easy to understand because the basic logic behind the story is easy and there are only two phases and three variables. But the actual implementation is hidden too deep, and the idea of Paxos is not enough to solve the consensus problems in the reality. Paxos has left many spaces for developers to explore, so there is no standard implementation of the algorithm. On the other hand, Raft is designed on the basis of Paxos with the purpose of defining a consensus algorithm for practical systems and to describe it in a way that is significantly easier to learn than Paxos (Ongaro & Ousterhour, 2013). Raft provided a clearer process and description to solve the consensus problems, which could be used directly for implementation. In Howard and Mortier's study in the difference between Paxos and Raft (Howard & Mortier,2020) one of the main differences is the implementation of leader election, Paxos would choose a candidate with the higher proposal ID as their leader. Therefore, theoretically any proposer could be chosen, so a log check is necessary to guarantee the leader's log is up to date, this could be time-consuming. Leader election in Raft ruled that they will

only elect candidates containing all committed entries and thus need not pass log entries again during the election. Even though situations such as votes splitting could occur because they have the same term, Raft's leader election is still more lightweight than Paxos. And from the perspective of safety, they are both strong consensus which means that they will have the same result in the end. Paxos uses proposer ID and Raft uses term to verify the consensus of the result. The value proposed from a server must be accepted and recorded by the majority of the other servers before it gets formally proposed. If another server wants to propose a new value, it will need to ask for the majority of the servers to get acceptance. Since the acceptors have received the message before this, they have the ability to make decisions either to reject it to propose or accept it. Either one can ensure the consistency will not be corrupt. Last, the ordering of log entries is different between these two algorithms. "suppose a leader can get α commands ahead—that is, it can propose commands i + 1 through i +α after commands 1 through i are chosen. A gap of up to α−1 commands could then arise." (Lamport,2001). Because the internet and message passing is not always reliable, there is a possibility of failure would occur during the process. Paxos allows its proposal ID to be out of order. Therefore, the existence of gap is acceptable in Paxos. Where in Raft, log entries are added in order, because followers will only append a log entry when its log is identical to the leader's log. Which ensures the correct log entries to their state machine (Howard & Mortier,2020).

V. Conclusion

In conclusion, consensus algorithms are gaining more attention because of the rapid development in distributed computing. In this paper, we have introduced the logic of Paxos and Raft algorithms and compared their differences.

Paxos and Raft as one of the first developed algorithms have been completed by many researchers, these algorithms have proved their efficiency and correctness.

Bibliography

A. Fox and E. Brewer, "Harvest, yield, and scalable tolerant systems," , 1999. [Online]. Available: https://cs.uwaterloo.ca/~brecht/servers/readings-new2/harvest-yield.pdf. [Accessed 7 6 2022].

Lamport, L. (2019). The part-time Parliament. *Concurrency: the Works of Leslie Lamport*. https://doi.org/10.1145/3335772.3335939

Boichat, R., Dutta, P., Frølund, S., & Guerraoui, R. (2003). Deconstructing Paxos. *ACM SIGACT News*, *34*(1), 47–67. https://doi.org/10.1145/637437.637447

Ongaro, D., & Ousterhout, J. (2013). In search of an understandable consensus algorithm (extended version).

Howard, H., & Mortier, R. (2020). Paxos vs raft. *Proceedings of the 7thWorkshop on Principles and Practice of Consistency for Distributed Data*. https://doi.org/10.1145/3380787.3393681

Lamport, L. (2006). Fast paxos. *Distributed Computing*, *19*(2), 79–103. https://doi.org/10.1007/s00446-006-0005-x

Lamport, L. (2001). Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, 51-58