CS5232 Research Project Proposal

Formalisation and Specification Design of Fast Paxos Algorithm using TLA+.

Lim Ngian Xin Terry (A0218430N)
Gaurav Gandhi (A0254394X)

Introduction

The systems distributed across different locations, communicating and interacting amongst themselves through message passing forms the backbone of the modern cloud infrastructure. It is really important for those systems to agree upon a particular value in order to maintain consistency across the systems. In order to achieve this consistency many different consensus algorithms have been proposed in past which handle certain particular scenarios really well but fail to maintain consistency under systems failure. Algorithms like Transaction Commit and Two-Phase Commit are examples of such algorithms. The Two-Phase Commit protocol uses a single coordinator to achieve consensus in a distributed systems environment, which fails to reach a consensus if the coordinator process is crashed or gets disconnected from the rest of the system [1].

This is where Paxos Algorithm and its different variants come into the picture. The original Paxos algorithm, often referred to as Classic Paxos or Vanilla Paxos was proposed by Leslie Lamport in the year 2001 [1] followed by its variant Fast Paxos in the year 2006 [4]. The Classic Paxos algorithm assigns different roles *(i.e., Proposer, Acceptor and Learner)* to the different systems in the distributed environment, where each role has a specific task to perform. The Classic Paxos algorithm usually has one *proposer*. There is also a variant of Classic Paxos in which the system can have more than one *proposer* in system, which is known as *Multi Paxos [2]*.

What is Fast Paxos?

Fast Paxos is an optimization of the original Paxos algorithm (referred to as Classic Paxos), which is further optimized to reduce the latency for arriving at a consensus in a distributed systems environment with the help of larger quorum sizes. A variety of fault-tolerant systems are implemented in real-world scenarios with the help of the Classic Paxos algorithm but a further reduction of latency is implemented in the Fast Paxos. Just like the Classic Paxos, Fast Paxos also operates in rounds and there are two phases in each round [4].

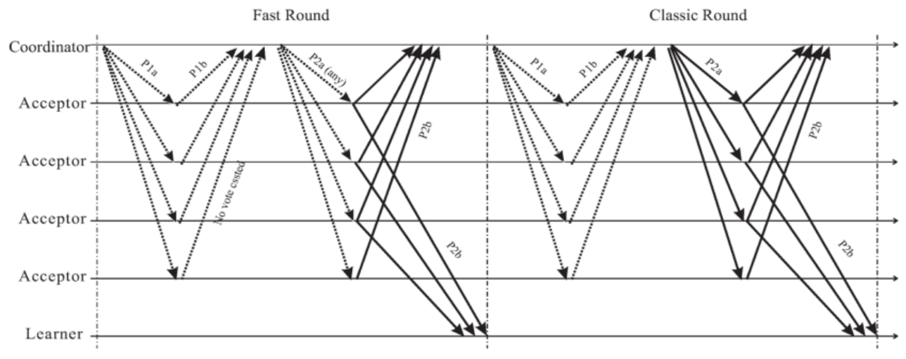

Figure 1. Each round in Fast Paxos comprises of two phases. There are two varieties of rounds in Fast Paxos: classic and fast. The coordinator selects the value to be voted in classic round whereas in the case of fast round each acceptor can propose its own value. [4]

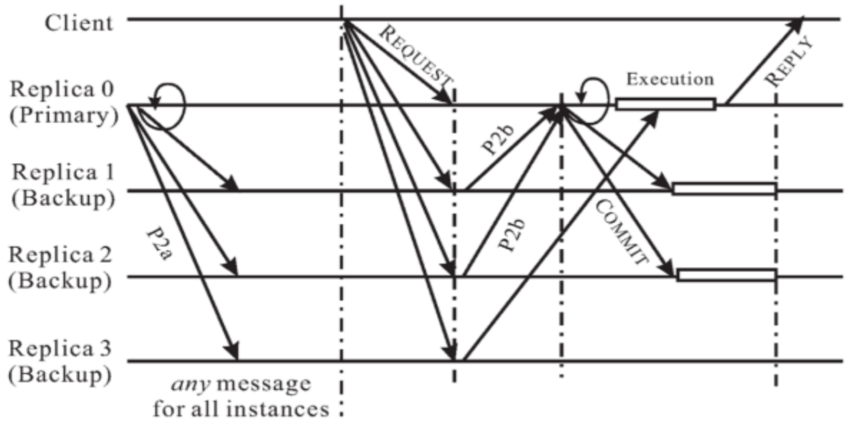

Figure 2. A normal operation of a (Multi) Fast Paxos in a client-server system [4]

It has 2 types of rounds, fast and classic. Each round consists of multiple phases: *preparation, acceptance and learning [5]*. When a sole coordinator is present, the preparation phase can be omitted.

Assume that a sole coordinator is present. As can be seen in *Figure 1* above, in the fast round, acceptors are allowed to propose values directly to the coordinator in the acceptance phase. The coordinator will then select a value if it has been proposed by a majority of a quorum of acceptors. This reduces latency compared to Classic Paxos as the coordinator does not need to propose a value.

If the coordinator receives the same value from a quorum of acceptors, a value is selected. If a collision exists and the coordinator is unable to select a value, it will initiate a classic round.

Once a value has been selected, it is propagated to all acceptors and learners in the learning phase.

Where can it be applied?

Fast Paxos is faster than Classic Paxos when there is no collision in which case only the fast round will be executed, but slower when there is a collision as in that case along with the fast round, a classic round is also performed. Therefore, Fast Paxos is most suitable for use in single-client configuration systems, as having two or more clients in the system will result in frequent collisions.

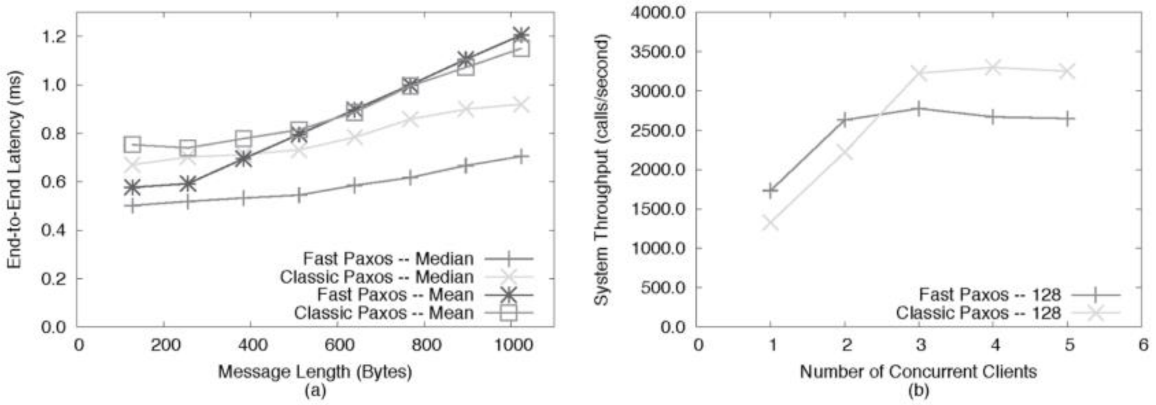

Figure 3. End-to-end latency (a) and throughput (b) for the replicated client server application. [4]

Examples of possible low-collision systems:

- Player data in online games
    - Players can only log in using one device at a time, and only connect to one server at a time.
- Online shopping cart

    - Users generally browse the marketplace using a single device at a time
    - Add-to-cart/Remove-from-cart events generally only occur once every few minutes, giving enough time for Fast Paxos to complete.

As there are many low-collision systems in the real world, Fast Paxos can be adopted for such distributed systems.

Why is it interesting?

Classic Paxos and Fast Paxos require different quorums with different failure tolerance. Let us suppose that there is n number of acceptors in a distributed systems environment.

- In Classic Paxos the quorum size is n/2 + 1, and (n-1)/2 faulty acceptors can be tolerated [4].
- In a Fast Paxos classic round, the quorum size is n/2 + 1, and (n-1)/2 faulty acceptors can be tolerated [4].
- In Fast Paxos fast round, the quorum size is 3n/4, and n/4 faulty acceptors can be tolerated [4].

The original Paxos algorithm does not need to have a coordinator, while Fast Paxos relies on a single Coordinator. In the presence of a coordinator, when there is no collision, Fast Paxos only needs to send the acknowledgement message, known as P2b, while Classic Paxos needs to send the proposed value message, known as P2a, as well as P2b. In the presence of a coordinator, Classic Paxos only has a sole proposer. Fast Paxos allows multiple proposers to propose the same or different values with the same or different ballot numbers.

Why is it feasible?

The Fast Paxos algorithm can be broken down into 3 specifications:

1. Classic Paxos w/o Coordinator
2. Classic Paxos w/ Coordinator (AKA Multi-Paxos)
3. Fast Paxos

Each specification is built upon the previous. Therefore, the task can be broken into 3 smaller, more manageable tasks.

Risks

Some possible risks may be the time needed for model checking to run if the number of replicas is too large. In such cases, we can limit the number of replicas to a maximum of 4, which is the minimum required for Fast Paxos to tolerate 1 faulty node.

Minimum Viable Product

A minimum viable product would be to complete the formalisation of the fast round of Fast Paxos.

Stretch Goals

The current specification we are proposing includes cases where the acceptors fail. For our stretch goal, we can try to include cases where the coordinator fails.

Implementation Timeline

Week	Milestone
6	Proposal
7	Homework 1 & TLA+ Familiarisation
8	Formalising Paxos
9	Formalising Multi-Paxos
10	Formalising Fast Paxos
11	Bug Fixes
12	Presentation
13	Presentation

References

1. Gray, J., & Lamport, L. (2006). Consensus on transaction commit. *ACM Transactions on Database Systems*, *31*(1), 133-160. https://doi.org/10.1145/1132863.1132867
2. Lamport, L. (2006). Fast Paxos. *Distributed Computing*, *19*(2), 79-103. https://doi.org/10.1007/s00446-006-0005-x
3. Luis Quesada Torres. (2018, March 1). *The Paxos Algorithm* [Video]. YouTube. https://www.youtube.com/watch?v=d7nAGI_NZPk
4. Zhao, W. (2015). Fast Paxos made easy. *International Journal of Distributed Systems and Technologies*, *6*(1), 15-33. https://doi.org/10.4018/ijdst.2015010102
5. Lamport, L. (2001). Paxos Made Simple. *Distributed Computing*