Fast Paxos Made Easy:
Theory and Implementation

Wenbing Zhao, Department of Electrical and Computer Engineering, Cleveland State University, Cleveland, OH, USA

ABSTRACT

Distributed consensus is one of the most important building blocks for distributed systems. Fast Paxos is one of the latest variants of the Paxos algorithm for distributed consensus. Fast Paxos allows an acceptor to cast a vote for a value of its choice unilaterally in a fast round, thereby eliminating a communication step for reaching consensus. As a tradeoff, the coordinator must build a quorum that is bigger than the simple majority used in Classic Paxos. This article presents the theory, implementation, and a comprehensive performance evaluation of the Fast Paxos algorithm. The theory is described in an easier-to-understand way compared with the original article by Lamport. In particular, an easy-to-implement value selection rule for the coordinator is derived. In the implementation of Fast Paxos for state-machine replication, a number of additional mechanisms are developed to cope with practical scenarios. Furthermore, the experiments reveal that Fast Paxos is most appropriate for use in a single-client configuration. The presence of two or more concurrent clients even in a local area network would incur frequent collisions, which would reduce the system throughput and increase the mean response time as experienced by clients. Due to frequent collisions, Fast Paxos actually performs worse than Classic Paxos in the presence of moderate to large number of concurrent clients.

Keywords: *Distributed Consensus, End-to-End Latency, Fault Tolerance, Probability Density Function, Quorum Requirements, State-Machine Replication, System Throughput*

INTRODUCTION

Distributed consensus is one of the most important building blocks for distributed systems (Zhao, 2014). For example, it is impossible to build a highly available cloud service without using some distributed consensus algorithm to ensure that all replicas remain consistent (Camargos, Madeira, & Pedone 2006; Camargos, Schmidt, & Pedone, 2008; Zhao, Melliar-Smith, & Moser, 2010; Zhao, 2010). Fast Paxos (Lamport, 2006) is one of the latest variants of the original Paxos algorithm (Lamport, 2001) (referred to as Classic Paxos) for distributed consensus. Classic Paxos is a good fit for state-machine replication and it has been used in a number of practical fault tolerant systems (Bolosky et al., 2011; Burrows, 2006; Hunt et al., 2010; Mao et al., 2008; Rao, Shekita, & Tata, 2011). Fast Paxos aims to further reduce the latency for reaching consensus by using a larger quorum size. Similar to Classic Paxos, Fast Paxos operates in rounds and there are two phases in each round. If a consensus is not reached within a round, a new round will be launched for liveness. In Fast Paxos, there can

DOI: 10.4018/ijdst.2015010102

be two different types of rounds: fast rounds and classic rounds. A classic round would operate the same way as a round in Classic Paxos except that the value selection rule at the coordinator is different, as to be explained in later sections. In the original article published by Lamport (Lamport, 2006), the quorum requirement as well as the value selection rule depend on the evaluation of the following observation known as O4(v) in (Lamport, 2006):

A value has been or might yet be chosen in round k only if there exists a k-quorum R such that vr(a) = k and vv(a) = v for every acceptor a in RTQ. (Lamport, 2006)

Here Q refers to the quorum formed for the current round, k is the most recent round number in which an acceptor a has casted a vote, k-quorum means the quorum used in round k, vr(a) refers to the round number in which the acceptor a has casted a vote, and vv(a) refers to the value contained in that vote. O4(v) is true if and only if the above observation is true for some round k for the value v.

As we can see, to evaluate this observation, one must examine every previous round k, and determine whether or not a k-quorum exists for round k that satisfies the specific constraint on v. This implies that for the coordinator to evaluate whether or not a value v satisfies O4(v), it must collect votes from every acceptor of the system in every round, which is simply not practical in asynchronous environment.

In this article, we introduce a more implementation-friendly value selection rule for the coordinator, and provide a more intuitive reasoning on the quorum requirements, both without the need to evaluate O4(v). To demonstrate the practicality of the proposed value selection rule, we present an implementation of Fast Paxos for state-machine replication. We show that many additional mechanisms are needed to cope with practical scenarios. Furthermore, we have conducted a comprehensive evaluation of Fast Paxos using our research prototype. Our experiments reveal that Fast Paxos is most appropriate for use in a single-client configuration. The presence of two or more concurrent clients even in a local area network would incur frequent collisions, which would reduce the system throughput and increase the mean response time as experienced by clients. Due to frequent collisions, Fast Paxos actually performs worse than Classic Paxos in the presence of moderate to large number of concurrent clients.

The remaining of the article is organized as follows. Section 2 describes the system model used in Fast Paxos as well as Classic Paxos and their variants. Section 3 defines the safety and liveness requirements for distributed consensus solutions. Sections 4 and 5 introduce Classic Paxos and its application in state-machine replication (referred to as Multi-Paxos) as the foundation for Fast Paxos. In Section 6, we describe Fast Paxos and our theoretical contributions. In Section 7 and Section 8, we report the details of our implementation and performance evaluation of Fast Paxos. We conclude the article with the final two sections on related work and concluding remarks.

SYSTEM MODEL

We consider a distributed system with a number of processes, and any one of them may propose a value. A process may participate in a distributed consensus algorithm (such as Classic Paxos and Fast Paxos) in one of three roles: (1) proposer, (2) acceptor, or (3) learner. A proposer is one that proposes values to be chosen and learned by others. An acceptor participates in agreement negotiation on the values proposed. A learner is one that learns the value that has been chosen. Note that the roles are logical and a process can assume multiple roles, e.g., a process may act both as a proposer and an acceptor.

In a client-server system, the clients send their requests to the server for processing and expect to receive the corresponding replies. When state-machine replication of the server is used, it is essential to ensure that all replicas deliver and execute the requests in the same total order. For each request, its total order

constitutes the value that requires a distributed consensus among the replicas. To determine the total order for each request, typically one of the replicas is designated as the primary and it is responsible to determine the total ordering. Hence, the value to be agreed upon is typically contributed by both the client and the primary (i.e., they jointly serve as a proposer). The primary is also referred to as the coordinator.

We assume that the system and the consensus algorithms operate in an asynchronous environment with crash faults only (i.e., no Byzantine faults). The asynchronous environment means that it may take a process arbitrary long time to complete a local task, and a message may take arbitrarily long time to be delivered at the intended destination process, possibly after many retransmissions. Furthermore, processes might fail and a failed process stops participating in the consensus algorithm, i.e., it crashes.

CORRECTNESS CRITERIA FOR DISTRIBUTED CONSENSUS

A sound consensus algorithm should ensure both the safety property and the liveness property defined below (Lamport, 2001):

- **Safety property:** The consensus algorithm should guarantee:
  - **(S.a):** If a value is chosen by a process, then the same value must be chosen by any other process that has chosen a value;
  - **(S.b):** The value chosen must have been proposed by one of the processes in the system;
  - **(S.c):** Only the value that has been chosen by some process can be learned by a process;
- **Liveness property:** Eventually, some value is chosen. Furthermore, if a value has been chosen, then a process in the system can eventually learn that value.

The safety requirement (S.a) ensures that the same value is chosen by all processes. The requirements (S.b) and (S.c) rule out trivial solutions, e.g., all processes choose a pre-defined value. The liveness property can be satisfied only during periods of system synchrony.

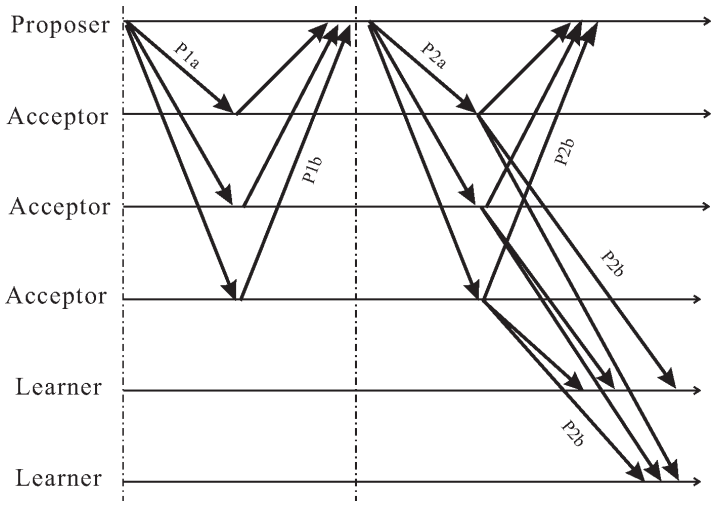
CLASSIC PAXOS

Classic Paxos (Lamport, 2001) operates in two phases, the prepare phase (i.e., phase one) and the accept phase (i.e., phase two), respectively, as shown in Figure 1. The prepare phase is initiated by a proposer sending a prepare request, referred to as P1a(n), to the acceptors in the system, where n is the proposal number selected by the proposer. At this stage, no value is included in the prepare request. This may appear to be counter-intuitive, but it is critical to limit the freedom of the proposer on what value it may propose because some acceptors might have accepted a value proposed by a competing proposer. Allowing a proposer to propose an arbitrary value at all times may lead to multiple values being accepted (as we will see, this limitation is lifted in Fast Paxos and mechanisms are defined so that at most one of the conflicting values is chosen in Fast Paxos (Lamport, 2006)).

In the prepare phase, when an acceptor receives a P1a(n) message, it does the following:

- If the acceptor has not responded to any P1a(n) message, it records the proposal number n, and sends its acknowledgment, referred to as P1b(n), to the proposer;
- If the acceptor has already responded to another P1a(m) message with a proposal number m, and $m < n$, there are two scenarios:
  - The acceptor has not received any accept request, which is sent by a proposer during the accept phase and referred to as P2a message, it records the higher proposal number n and sends its P1b(n) message to the proposer;
  - The acceptor has already received a P2a(k) message with a proposal num-

Figure 1. Normal operation of the Paxos algorithm



ber k, it must have received a value proposed by some proposer in the past. This full proposal [k, v] is included in P1b(n, [k, v]) to the proposer. Obviously, k must be smaller than n.

The second phase (i.e., the accept phase) starts when the proposer could manage to collect responses from the majority of acceptors. The proposer determines the value to be included in the P2a message in the following way:

- If the proposer received one or more P1b messages with full proposals. it selects the value v in the proposal that has the highest proposal number;
- If none of the P1b messages received by the proposer contains a full proposal, the proposer has freedom to propose any value.

Once the value is selected, the proposer multicasts a P2a(n, v) message to the acceptors. When an acceptor receives a P2a(n, v) message, it accepts the proposal [n, v] only if it has responded to the corresponding P1a(n) message. Then, it sends an acknowledgment message, referred to as P2b(n).

Note that when an acceptor accepts a P2a message, it does not mean that the value contained in the proposal included in P2a has been chosen. Only after the majority of acceptors have accepted the same P2a does the value is considered chosen. It is possible that no value is chosen or another value is eventually chosen after a minority of acceptors have accepted a P2a message.

There are a number of ways for a learner to find out the value that has been chosen. The most straightforward method is for an acceptor to multicast P2b to all learners, as shown in Figure 1. When a learner has collected P2b messages for the same proposal from the majority of acceptors, it will be rest assured that the value has been chosen. As an alternative, if the number of learners is large, a small group of learners can be selected to receive the multicasts from the acceptors and they can relay the accepted value to the remaining learners. Yet another alternative is for each learner to periodically poll the acceptors to see if they have accepted a value.

If a learner wants to make sure that the value it has learned is indeed the value that has been chosen, it can ask a proposer to issue a

new proposal. The result of this proposal would confirm whether or not the value is chosen.

MULTI-PAXOS

An immediate application of the Classic Paxos algorithm is to enable state-machine replication. As mentioned before, the value to be agreed on by the server replicas (i.e., acceptors) is the total order of the requests sent by the clients. The total ordering of a sequence of requests is accomplished by running a sequence of instances of the Classic Paxos algorithm. Each instance is assigned a sequence number, representing the total ordering of the request that is chosen. For each instance, the value to be chosen is the particular request that should be assigned to this instance.

The proposer in each instance is referred to as the coordinator (Lamport & Massa, 2004), the leader (Lamport, 2001), or simply the primary (Zhao, Zhang, Chai, 2009). The replica that serves as the proposer also acts as an acceptor. In a simple implementation, the primary propagates the chosen value to the remaining replicas (often referred to as backups) so that they can learn the value as well (Zhao, 2007; Zhao, Zhang, & Chai, 2009). Obviously, the primary would be the first to know that a value is chosen for each instance of the Classic Paxos algorithm, and usually the first to send the reply to the client. The backups can suppress their replies unless they have suspected the primary because the client needs only a single reply for each of its requests. It is also possible to enable a backup to learn the chosen value faster by multicasting each replica's P2b message to all replicas (instead of only to the primary). A trade-off for this approach is more multicast messages being sent in the system. Furthermore, a backup might learn the chosen value ahead of the primary.

Normally, one of the server replicas is designated as the primary at the beginning of the system deployment. Only when the primary becomes faulty, which is rare, or being suspected of being faulty by other replicas, another replica will be elected as the new primary. As long as there is a sole primary in the system, it is guaranteed that no replica would report having accepted any proposal to the primary, which would enable the primary to select any value (i.e., assigning any request to the current instance). Therefore, the first phase (i.e., the prepare phase) can be omitted during normal operation (i.e., when there is only a single primary in the system).

The full Classic Paxos algorithm is needed to elect a new primary. Furthermore, this run would effectively execute the first phase of all instances of the Classic Paxos as long as the current primary is operating correctly.

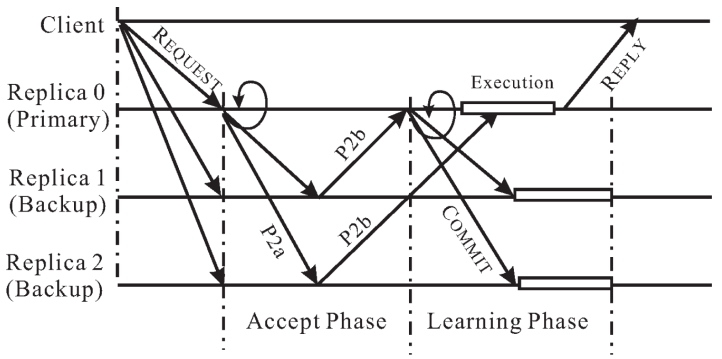
The above scheme of applying the Classic Paxos algorithm for state-machine replication is first proposed in (Lamport, 2001) and the term "Multi-Paxos" was first introduced in (Chandra, Griesemer, Redstone, 2007). TheMulti-Paxos algorithm during normal operation is illustrated in Figure 2. Note that the primary can execute the request as soon as it receives the P2b messages from a quorum of replicas.

FAST PAXOS

The objective of Fast Paxos (Lamport, 2006) is to reduce the end-to-end latency of reaching a consensus in scenarios where the clients are responsible to propose values to be chosen by the acceptors. In Multi-Paxos, we have shown that the first phase of Classic Paxos can be run once for all instances of the algorithm provided that initially there is a single leader. Hence, in Multi-Paxos, the cost of reaching agreement is the second phase of the Classic Paxos algorithm. Fast Paxos aims to further reduce the cost of reaching consensus by enabling the running of one P2a message for all instances of Fast Paxos in state-machine replication. This would enable an acceptor to select a value (provided by a client) unilaterally and sends the P2b message to the leader immediately, thereby reducing the end-to-end latency.

Because the Classic Paxos algorithm is proven to be optimal (Lamport, 2001)1, to re-

Figure 2. Normal operation of Multi-Paxos in a client-server system with 3 server replicas and a single client



duce the latency, we must sacrifice something else. In Fast Paxos, to tolerate f faulty replicas, more than 2f + 1 replicas are required. We will develop the criteria on the minimum number of replicas to tolerate f faults for Fast Paxos to work. Furthermore, because an acceptor (i.e., a server replica) unilaterally selects a value (i.e., a request message sent by a client), different acceptors might select different values. This scenario is referred to as a collision (in choosing a value) in (Lamport, 2006). Collision avoidance and collision recovery are new problems that occur in Fast Paxos.

We first describe the basic steps of the Fast Paxos algorithm, then we discuss the quorum requirement and the value selection rule for the coordinator. We conclude the section by providing a proof of correctness of Fast Paxos with our modifications.

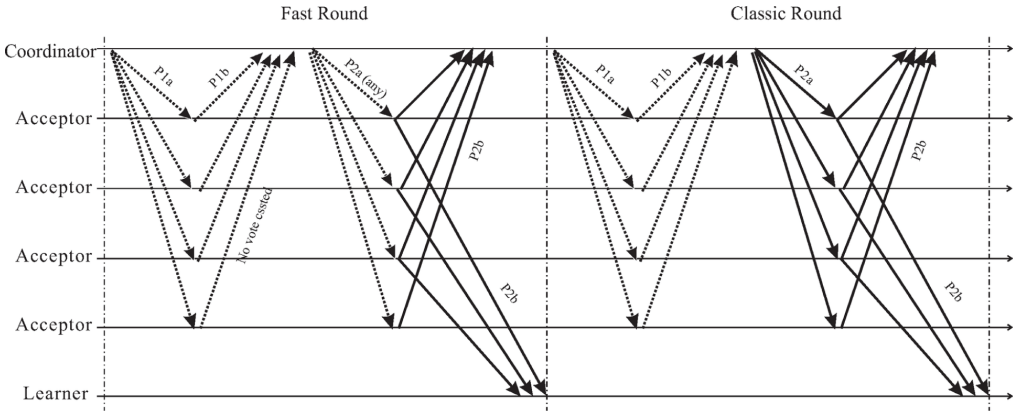
THE BASIC STEPS

Similar to Classic Paxos, Fast Paxos also operates in rounds (the round number corresponds to the proposal number in Classic Paxos) and each round has two phases, as shown in Figure 3. The first phase is a prepare phase to enable the coordinator to solicit the status and promises from the acceptors. The second phase is for the coordinator to select a value and be voted on by the acceptors. When an acceptor has responded to a P1a message in a round i, it is said that the acceptor has participated the round i. When an acceptor has sent to the coordinator a P2b message in response to the P2a message from the coordinator, it is said that the acceptor has casted its vote for that round. When the coordinator has collected P2b messages with the same value from a quorum of acceptors in that round, that value is said to have been chosen.

However, Fast Paxos has a number of differences from Classic Paxos:

- In Fast Paxos, a round may be either a fast round or a classic round. A fast round may use a quorum of different size than that of a classic round. We refer to the quorum used in a fast round as fast quorum, and the quorum used in a classic round as classic quorum;
- The value selection rule at the coordinator is different from that of the Classic Paxos due to the presence of the fast round;
- In a classic round, the coordinator selects the value to be voted on, similar to that of Classic Paxos;
- In a fast round, if the value selection rule allows the coordinator to select its own value, it may send a special P2a message to the acceptors without any value selected.

Figure 3. Fast Paxos operates in rounds with each round consisting of two phases. A round can be a classic round, where the coordinator selects a value to be voted on, or a fast round, where each acceptor is allowed to propose its own value. The dotted arrowed lines means that they can be omitted when a unique coordinator exists in the system.



This special P2a message (referred to as any message in (Lamport, 2006)) enables an acceptor to select its own value (proposed by a client) to vote on.

A learner can learn the value that has been chosen using any of the learning mechanisms we outlined for Classic Paxos with one modification: instead of collecting from a majority of the acceptors to learn a value that has been chosen, the learner must collect from a classic quorum of acceptors in a classic round, and from a fast quorum of acceptors in a fast round.

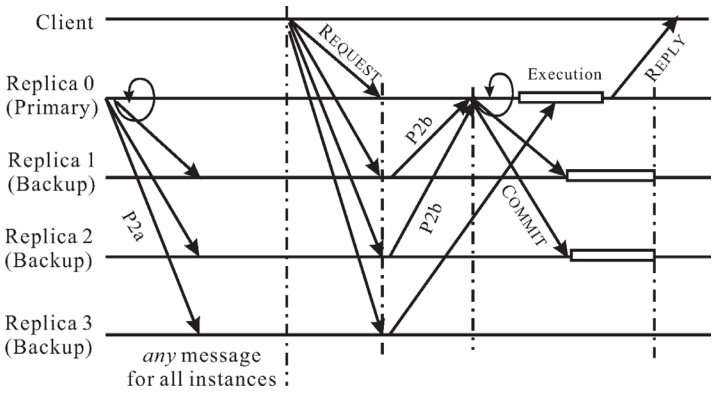
Assuming that there has been a unique coordinator since the server started running, the first time a fast round is run will always allow the coordinator to send an any message in phase 2. In a typical state-machine replicated system, this would allow the running of a single P2a message for all instances of Fast Paxos, which would eliminate one communication step, as shown in Figure 4. This is the sole advantage of Fast Paxos. Hence, whenever possible, a fast round is run and a classic round is used only when a consensus cannot be reached in the fast round due to the failure of the coordinator or due to a collision.

COLLISION RECOVERY, QUORUM REQUIREMENT, AND VALUE SELECTION RULE

During a fast round, if the coordinator issues an any P2a message, the acceptors would have freedom to select its only values. If there are several clients proposing different values concurrently (i.e., they issue requests to the server replicas concurrently), it is likely that different acceptors could select different values, which would cause a collision. When this happens, the coordinator might see different values in the quorum of votes it has collected, which would prevent the consensus from being accomplished in this fast round.

Note that it is not an option for the coordinator to block waiting until it has collected votes with the same value from a quorum of acceptors because it may never be able to build a quorum if less than a quorum of acceptors have voted for the same value. Therefore, on detecting a collision, the coordinator should initiate recovery by starting a new, classic round. In this new classic round, it is apparent that the coordinator would receive the same, or similar information from a quorum of acceptors in the

Figure 4. Normal operation of (Multi-) Fast Paxos in a client-server system



first phase of the new round. Therefore, the first phase can be omitted and the coordinator can proceed to determining a value to be voted on in the second phase.

With a quorum of votes containing different values, the coordinator must be careful in selecting a value that has been chosen in a previous round, or might be chosen. Just like Classic Paxos, Fast Paxos does not terminate, and hence, once a value is chosen, the same value must also be chosen in any future round. A value is chosen or might be chosen if a quorum of acceptors have voted the same value. Choosing any other value might cause two or more values be chosen, which would violate the safety property for consensus. However, it is not straightforward for the coordinator to determine if a value in the quorum of votes has been chosen or might be chosen.

Before we delve further on the value selection rule, we first show that the simple-majority based quorum formation in Classic Paxos is no longer valid in Fast Paxos. In Classic Paxos, to tolerate f faulty acceptors, a total of 2f + 1 acceptors are required and the quorum size is a simple majority (f + 1). With a quorum size of f + 1, two quorums may intersect in as few as a single acceptor. Therefore, with this quorum formation, a coordinator cannot rule out the possibility that a value might have been chosen even if it has collected a single vote with that value. As such, the coordinator would not be able to determine which value to select if it sees different values in the quorum of votes it has collected. Note that only one of the different values could have been chosen because it is impossible for the acceptors to form two quorums each with a different value in the same round even if a quorum is formed by a simple majority. The problem then becomes to determine which of the values is the most likely candidate such that if a value has been chosen in a previous round, that value is guaranteed to be selected.

It should be apparent that a bigger quorum than the simple majority must be used in Fast Paxos. The most intuitive way for the coordinator to determine whether or not a value has been chosen is to see whether or not there is a clear majority of votes for a common value in the current quorum. Note that the presence of a common value from the majority of votes in a quorum does not necessarily mean that the value has been chosen. Our next task is to ensure that a value in any of the minority votes could not have been chosen in the past. We can guarantee this property by imposing the following quorum requirement in addition to the basic quorum requirement:

- A fast quorum Rf and a classic quorum Rc must intersect in more than |Rc|/2 acceptors.

Therefore, we have the following three quorum requirements:

1. Any two classic quorums must intersect in at least one acceptor;
2. Any two fast quorums must intersect in at least one acceptor;
3. Any fast quorum Rf (with a size |Rf |) and any classic quorum Rc (with a size |Rc|) must intersect in more than |Rc|/2 acceptors.

With the list of quorum requirements in place, we are now ready to derive the quorum sizes. Let the total number of acceptors be n, the number of faulty acceptors that can be tolerated in a classic round be f, and the number of faulty acceptors that can be tolerated in a fast round be e. Hence, a classic quorum is formed by $n - f$ acceptors and a fast quorum is formed by $n - e$ acceptors. Based on our previous argument, it is clear that $f \geq e$. The three quorum requirements are translated to the following:

$$(n - f) + (n - f) - n > 0$$
$$(n - e) + (n - e) - n > 0$$
$$(n - f) + (n - e) - n > (n - f)/2$$

The requirements can be further reduced to:

$$n > 2f$$
$$n > 2e$$
$$n > 2e + f$$

As we can see, the quorum requirement (2) is superseded by the quorum requirement (3) because the latter is more restrictive. We end up with only the following two quorum requirements:

$$n > 2f \quad (1)$$

$$n > 2e + f \quad (2)$$

We can have two different quorum formations by maximizing e or f:

- **First quorum formation:** Because $f \geq e$, to maximize e, we have $e = f$ and $n > 3f$. Hence, a classic quorum would be the same size of a fast quorum: $|Rc| = n - f > 3f - f = 2f$. For all practical purposes, the total number of acceptors would be set to $n = 3f + 1$ and the quorum size (both classic and fast) would be $2f + 1$. For example, if we choose $f = 1$, we would need a total of 4 acceptors, and the quorum size would be 3;
- **Second quorum formation:** To maximize f, we can use the upper bound given in Equation 1 for f, therefore:

$$f < n/2$$

We can derive the requirement on e from Equation 2:

$$e < (n - f)/2$$

By replacing f with n/2 (i.e., f's upper bound)), we have:

$$e \leq (n - n/2)/2$$

Finally, we have:

$$e \leq n/4$$

Therefore, the size of a classic quorum must be greater than n/2 (i.e., a simple majority), and the size of a fast quorum must be greater than 3n/4. For example, if we use the smallest e possible, i.e., $e = 1$, we need a minimum of 4 acceptors. The size of a fast quorum would happen to be the same as that of a classic quorum, which is 3. Note that $f = 1$ as well. Furthermore, a classic quorum does not always have the same size as that of a fast quorum. Consider the case when $e = 2$. We would need to have 8 acceptors, which means that a classic quorum must consist of 5 acceptors while we would need 6 acceptors to form a fast quorum. Hence, $f = 3$ in this case.

Having fully defined the classic and fast quorums for Fast Paxos, it is time to define the value selection rule at the coordinator. We have already argued that in case of different values are present in the votes that the coordinator has collected, the coordinator should choose the value contained in the majority of the votes in the (classic) quorum, if such a value exists. If no such majority votes exist in the quorum, the coordinator is free to choose any value because no value could have been chosen in a previous round due to our quorum requirement 3. Hence, the value selection rule is defined below:

1. If no acceptor has casted any vote, then the coordinator is free to select any value for phase 2;
2. If at most a single value is present in the votes, then the coordinator must select that value;
3. If the votes contain different values, a value must be selected if the majority of acceptors in the quorum have casted a vote for that value. Otherwise, the coordinator is free to select any value.

Rule 1 and rule 2 are the same as those for Classic Paxos. The rule 3 is specific for Fast Paxos. Compared with the original coordinator's rule in (Lamport, 2006), our value selection rule for the coordinator has the following characteristics in cases of collisions:

- If according to the original coordinator's rule, a value is selected, the same value is guaranteed to be selected according to our rule. This is because if a value has been chosen previously, according to our quorum requirement, that value must be present in the majority of votes;
- However, a value that is selected according to our rule might not have been chosen previously, in which case, the coordinator would have the freedom to select any value according to the original rule. Note that our rule in this respect does not have any negative impact to the consensus algorithm: if the coordinator has freedom to select any value, it sure is allowed to select the one that is present in the majority of votes.

PROOF OF CORRECTNESS

We prove that the modified Fast Paxos with our value selection rules satisfies the safety properties outlined previously. We choose not to prove the liveness property because our modifications to Fast Paxos is not related to liveness:

Theorem 1: Safety property S.a: If a value is chosen by a process, then thecsame value must be chosen by any other process that has chosen a value.

Proof: We prove by contradiction that S.a is satisfied by Fast Paxos. Assume value v is chosen in some round t, and another value u is chosen in some other round s. Without loss of generality, we assume $t < s$. From round $t + 1$ to round s, the coordinator for each round must collect information from a quorum of acceptors to select the value to be voted on in phase 2. If between two consecutive rounds r and r+1, the coordinator is not changed, the coordinator obtains such information via the P2b messages in round r. If the coordinator is changed between round r and r+1, the new coordinator must collect P1b messages in the phase 1 of round r+1.

There are only four scenarios for consecutive rounds:

- Round r is a fast round and round $r + 1$ is also a fast round;
- Round r is a fast round and round $r + 1$ is a classic round;
- Round r is a classic round and round $r + 1$ is a fast round;
- Round r is a classic round and round $r + 1$ is also a classic round.

We start by considering the case when r is t, where a value v is chosen. Furthermore, we assume that the coordinator might have been changed from round t to round t + 1. Therefore, phase 1 in round t + 1 might be necessary. Under the four scenarios we outlined above.

A fast quorum R1f of acceptors has voted for v in round t. In round t + 1, the coordinator would collect information from a fast quorum R2f of replicas to compute the value to be selected for P2a. The size of two fast quorums is n − e (where n is the total number of acceptors and e is the number of faulty acceptors tolerated). R1f and R2f must intersect in a set S of at least 2n−2e−n = n−2e acceptors. According to the first quorum formation, f = e, and n = 3f + 1. Hence, |R1f | = |R2f | = 2f + 1, and |S| = f + 1. According to the second quorum formation, n = 4e + 1. Hence, |R1f | = |R2f | = 4e + 1, and |S| = 2e + 1. It is clear that for both quorum formations, a majority of the acceptors in R2 f are also in R1f. According to our value selection rule, the coordinator must select v in round t+1 for voting, if the coordinator could manage to complete the first phase in round t+1. Furthermore, if the coordinator could manage to finish the second phase, v must to be chosen in round t + 1.

In scenario 2, the coordinator in round t + 1 collects information from a classic quorum Rc acceptors. The fast quorum R1f of acceptors that voted v in round t and the classic quorum Rc must intersect in at least (n−f)+(n − e) − n = n − f − e > (n − f)/2 acceptors due to Equation 2, which is the majority of the classic quorum of acceptors in round t+1. This ensures that v will be selected if the coordinator could complete the first phase in round t+1. Similar to scenario 1, if the coordinator could manage to finish the second phase, v must to be chosen in round t + 1.

In scenario 3, because v is chosen in a classic round t, the operation in this round is reduced to Classic Paxos, which means that only v could have been voted by any acceptor. Hence, in round t + 1, the coordinator would only see a single value v among the votes it collects. According to the value selection rule, the coordinator must select v in round t+1 if the coordinator could collect information from a quorum of acceptors. Furthermore, if the coordinator could manage to finish the second phase, v must to be chosen in round t + 1.

Due to the same reason, in scenario 4, the coordinator for round t+1 must also select v for voting in phase 2 if the coordinator could collect information from a quorum of acceptors. Furthermore, if the coordinator could manage to finish the second phase, v must to be chosen in round t + 1.

Therefore, we have the same conclusion for each scenario, i.e., the coordinator in the new round must select v for voting if it could manage to complete the first phase, and must chose v if it could complete the second phase of the new round. This conclusion is true for each new round until round s, which is conflicting with our assumption that a different value u is chosen in round s. This proves that the safety property S.a holds:

Theorem 2: Safety property S.b: The value chosen must have been proposed by one of the processes in the system.

Proof: In Fast Paxos, the value chosen is always selected by the coordinator according to the value selection rule. The value selected is proposed either by a proposer or one of the acceptor in a fast round. Hence, the safety property S.b holds.

Theorem 3: Safety property S.c: Only the value that has been chosen by some process can be learned by a process.

Proof: Any of the learning mechanisms we outlined for Fast Paxos can be used to learn the value chosen for Fast Paxos. It is easy see that the safety property S.c is satisfied trivially by any of the learning mechanisms.

IMPLEMENTATION

We implemented the Fast Paxos algorithm as part of a state-machine replication framework using the Java programming language. In the framework, Fast Paxos is used to ensure the

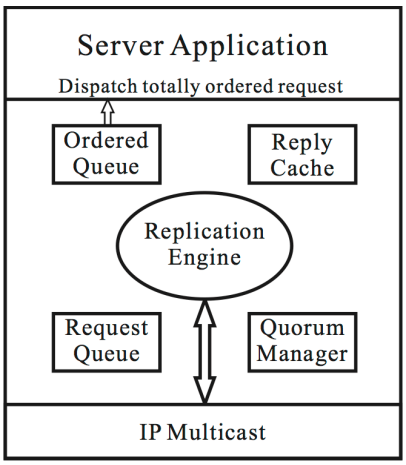
total ordering of requests for execution. The major components of the framework are shown in Figure 5.

The framework is designed for use in a local area network (LAN) where the client and the server replicas communicate via IP multicast. All incoming messages and outgoing messages at the server replica are controlled by the Replication Engine logic. There are two types of messages: (1) application messages (requests and replies), and (2) control messages used for total ordering of application requests such as P2a and P2b messages. A new application request is placed in the Request Queue immediately after received. When a replica is ready to order a request, it is removed from the Request Queue and placed in the Ordered Queue at the designated place. When an agreement on the total ordering of a request has been achieved using Fast Paxos, the request is dispatched to the server application for execution. A copy of the reply message resulted from the execution is stored in the Reply Cache for possible retransmission before it is sent to the client. Because Fast Paxos is a quorum-based algorithm, a Quorum Manager is used to keep track of the status of the quorum building process by the primary replica (i.e., the primary serves both the coordinator and the acceptor roles as defined in Fast Paxos). The Quorum Manager is not used at the backup replicas.

The Replication Engine implements Fast Paxos and drives all operations. An event driven approach is used in our implementation where all events are handled by an event loop. The event loop blocks waiting for a new message from the network (i.e., waiting for the next event) at the beginning of each iteration, and invokes the appropriate event handler for the message received according to its type. There are four event handlers:

- **Request handler:** It hands application requests sent by a client;
- **P2a handler:** It handles P2a messages. It is used only by backup replicas (i.e., acceptors);
- **P2b handler:** It handles P2b messages. It is used only by the primary replica;
- **Learn notification handler:** It handles the notification message (which we refer to as the Learn message) sent by the primary regarding what value is chosen (i.e., the total ordering for a request). The handler is used only by backups.

Figure 5. Major components of the state-machine replication framework with Fast Paxos



REQUEST HANDLER OPERATIONS

The request message is first checked to see if it is a duplicate. Duplicate requests are discarded. The Reply Cache is searched to see if a reply has been generated for the retransmitted request. If one is found, the reply is retransmitted. No further operation is carried out in this case.

Next, the request is checked to see if it is one of missing messages that the replica needs to execute. If it is, the message is placed in the Ordered Queue at the designated position for execution, all eligible ordered requests will be executed in the total order imposed by Fast Paxos.

If the received request is indeed a new request, it is appended to the Request Queue. The remaining operations depend on whether or not the replica is the primary or a backup, and whether or not it is in a fast round or it is in a classic round.

In a classic round, if the replica is the primary, a P2a message is prepared and multicast to all backups. The P2a message includes the next sequence number for the current request (which decides on the total ordering of the request). The request is also transferred from the Request Queue to the Ordered Queue. The primary then waits for the corresponding P2b messages from a quorum of replicas via the Quorum Manager. If the replica is a backup, it takes no further action (other than adding the request to the Request Queue).

In a fast round, both the primary and a backup replica order the request and prepare the corresponding P2b message independently. The P2b message carries the sequence number for the request and the request id. At the primary, the Quorum Manager creates a Certificate object for the corresponding sequence number and adds the P2b message prepared to the Certificate object. At a backup replica, the P2b message is sent to the primary.

P2A HANDLER OPERATIONS

Only a backup replica (i.e., an acceptor) receives the P2a message sent by the primary. First, the P2a message is checked to see if it is valid based on the sequence number included in P2a. If it is, the P2a message is handled according to the following mechanisms.

If it is in a classic round, the request being ordered is transferred from the Request Queue to the Ordered Queue, and the corresponding P2b message is prepared and sent to the primary.

If it is in a fast round, the very fact that a backup received a P2a message means that a collision has occurred and the primary is recovering the collision via the P2a message. In this case, the replica rolls back its choice of request for the corresponding sequence number by transferring the request from the Ordered Queue back to the Request Queue, and proceed to operating as if it is in a classic round by ordering the next request exactly as indicated in the P2a message, and responds with a P2b message to the primary.

P2B HANDLER OPERATIONS

The P2b message is only handled by the primary. The primary uses the Quorum Manager to build a quorum, and to select the value for a round of Fast Paxos using the value selection rule we described previously. If no collision is found (i.e., all replicas assigned the same request for the same sequence number), the primary proceeds to preparing and multicasting a Learn message to all backup replicas, informing them the chosen total ordering for a sequence number.

If a collision is detected, i.e., two or more different requests are selected for the same sequence number by different replicas, the primary selects at most one of them according to the value selection rule, and includes that value in the P2a message. To recover from the collision, the primary initiates a classic round by multicasting the P2a to backup replicas.

For a classic round, it is guaranteed that a collision will never happen. Normally, the system is configured to start with a fast round to enjoy the benefits of Fast Paxos and remain operating in fast rounds until a collision is detected. The primary uses a single classic round to recover from the collision. Once the classic round is over, the primary switches to fast rounds. The primary performs the switch-over when it is in a classic round and the quorum of P2b is complete.

LEARN NOTIFICATION HANDLER OPERATIONS

Only a backup replica may receive and handle a learn notification (i.e., Learn message). Upon receiving a Learn message, the replica knows that an agreement for a particular sequence number has been reached, and executes the ordered request in exactly the same order. Such a message also helps a slow replica catch up with other replicas by skipping the corresponding round of Fast Paxos consensus. To reduce unnecessary network load, the reply generated by a backup replica is suppressed.

When a backup replica receives a Learn message and it is in a classic round, it may infer that the collision recovery is over and switches to the fast round operation.

PERFORMANCE EVALUATION

The performance of Fast Paxos is evaluated with our state-machine replication framework, using Classic Paxos as a reference for comparison. The evaluation is carried out using a testbed consisting of five compact computers connected via a Gigabit Ethernet. Each computer node is equipped with a Core i5-4250U CPU and 4GB of RAM, and runs the Ubuntu 14.04 Linux. Four of the nodes are used to run the server replicas, and the remaining one is used to run the clients. For Fast Paxos, four replicas are used to tolerate a single faulty replica. For Classic Paxos, three replicas are used to tolerate a single faulty replica. A single client-server application is used to benchmark the system's performance. The server application is intentionally designed to perform no substantial processing for each request so that the end-to-end latency measured reflects primarily the communication and Fast Paxos consensus cost. The reply length is always set to be the same as the request length.

END-TO-END LATENCY

Because Fast Paxos is designed to eliminate one communication step in reaching a consensus, we expect that a major benefit of using Fast Paxos in state-machine replication is a reduced end-to-end latency as seen by a client. The end-to-end latency is defined to be the time elapsed since the client issues a request, until it receives the corresponding reply. For the end-to-end latency evaluation, we use a single client with varying request/reply message lengths. We keep track of the round-trip time of each request and save the data at the end of each run. The data is then used to compute the median, mean, as well as the probability density function of the end-to-end latency.

As shown in Figure 6(a), the experimental results are consistent with our expectation for requests with short lengths (i.e., 128 bytes and 256 bytes). However, the mean end-to-end latency for Fast Paxos is rather similar to that for Classic Paxos for longer requests. For a request with 1024 bytes, the mean latency for Fast Paxos is even slightly higher than that for Classic Paxos. This observation is rather puzzling initially. A closer look at the data shows that the median end-to-end latency for Fast Paxos is consistently smaller than that for Classic Paxos for all requests with different lengths (by about 25%, which reflects the benefits of eliminating one communication step).

The unexpected large mean latency for longer requests for Fast Paxos is apparently caused by a small fraction of data with large end-to-end latency. This is indeed the case, as illustrated by the probability density function of the end-to-end latency in Figure 7. For both Fast Paxos and Classic Paxos, we can notice the

Figure 6. End-to-end latency (a) and throughput (b) for the replicated client-server application

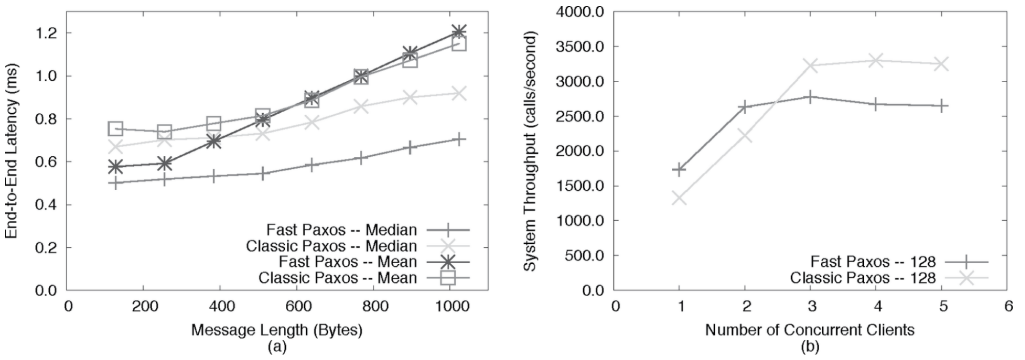
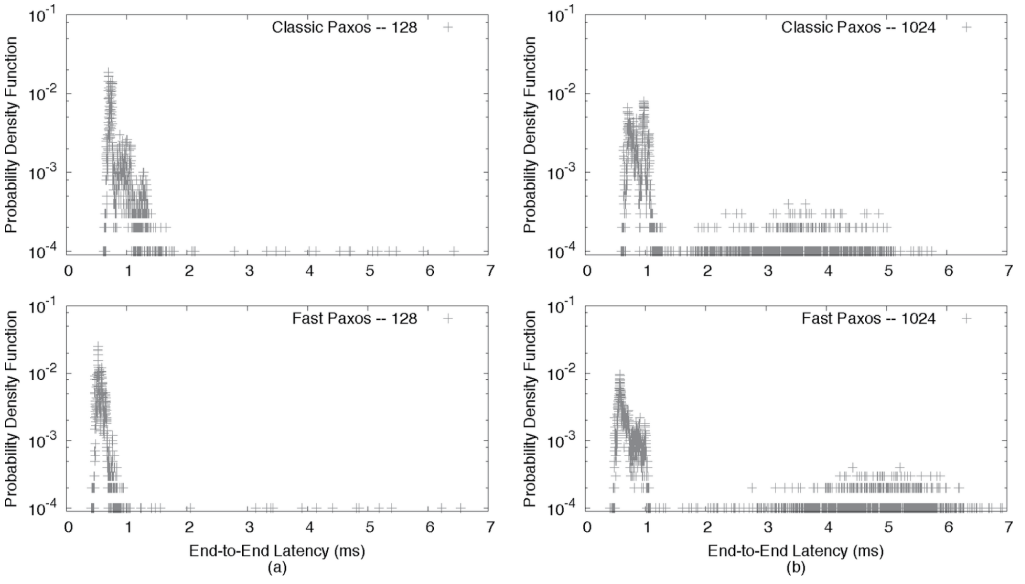


Figure 7. Probability density function of the end-to-end latency for Fast Paxos and Classic Paxos, for requests of 128-bytes long (a), and for requests of 1024-bytes long (b)



presence of the second broad peak for requests of 1024-bytes long. However, the second peak is positioned in larger end-to-end latency for Fast Paxos than that for Classic Paxos. The second peak is not present for requests of 128-bytes long. We speculate that this artifact is due to inefficiency of our implementation, rather than anything intrinsic to Fast Paxos. Future work is needed to identify the source of the problem.

SYSTEM THROUGHPUT

The system throughput is measured at the primary and it inevitably reflects the aggregated performance over a number of requests. To evaluate system throughput, we launch up to 5 concurrent clients with each client issuing requests of 128-bytes long consecutively. As shown in Figure 6(b), Fast Paxos has slightly

better system throughput than Classic Paxos in the presence of a single client or two concurrent clients. However, Fast Paxos performs worse in the presence of three or more concurrent clients. Other than the computational costs of building and processing a larger quorum compared against Classic Paxos, the main reason for the lower system throughput is frequent collisions and the ensuing recoveries. We observe that the collision rate is approximately 5% in the presence of concurrent clients. Hence, we conclude that Fast Paxos is not a good fit for high-throughput systems that must support large number of concurrent clients.

COLLISION RECOVERY LATENCY

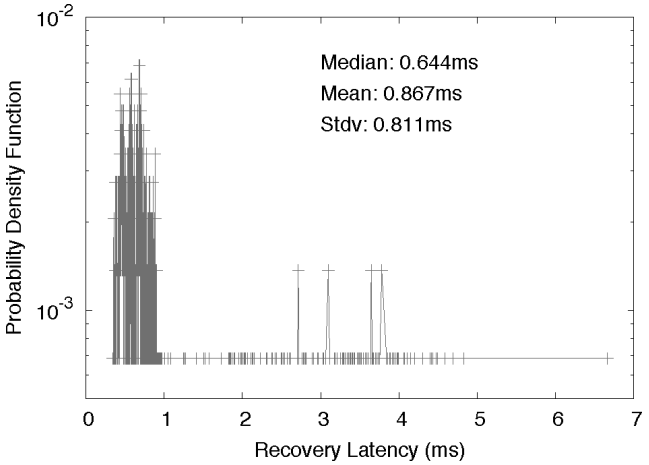
The collision recovery latency is measured as the time elapsed since the detection of a collision until the ensuring classic round is completed. Shown in Figure 8 is the probability density function of the recovery latency for a run as part of our throughput measurement where the number of clients varies from 1 to 5 (with a total of 1460 collision recoveries). As can be seen, there is a major peak close to 0.6ms and there exists a long tail extending as large as nearly 7ms. From Figure 8, we infer that the recovery latency distribution is relatively independent from the number of concurrent clients. This is expected because the recovery involves a single classic round regardless of the number of concurrent clients and the message lengths for P2a and P2b remain constant.

RELATED WORK

The impracticality of the coordinator's value selection rule is pointed out in (Vieira & Buzato, 2008). Not surprisingly, the rule derived in (Vieira & Buzato, 2008) is identical to ours. However, in (Vieira & Buzato, 2008), the rule is derived based on the quorum formation rule in (Lamport, 2006) (which is in turned derived from the requirement on O4(v)). In this article we essentially use this rule as the foundation to derive the quorum requirements without relying on the evaluation of O4(v). Furthermore, we recognize that the impracticality of the coordinator's rule is due to the difficulty of evaluating O4(v) for every possible value v in practical scenarios.

In (Junqueira, Mao, Marzullo, 2007), the performance of the Fast Paxos algorithm is compared against that of the Classic Paxos

Figure 8. Probability density function of the recovery latency



algorithm *via simulation* instead of an actual implementation, for wide area networks. The authors discovered that under certain circumstances, Classic Paxos outperforms Fast Paxos. Furthermore, the study is done for cases in which collision is absent in Fast Paxos. Since this study is based on simulation and it is framed for wide area networks, the result is not directly comparable to ours.

In (Vieira & Bzato, 2013), the performance comparison between Classic Paxos and Fast Paxos is studied. The conclusion is rather similar to that in (Junqueira, Mao, Marzullo, 2007) in that Classic Paxos outperforms Fast Paxos when the replication degree is small with and without collision in Fast Paxos. With large replication degrees, the performance of Fast Paxos and Classic Paxos are similar. This result is not surprising in that multiple concurrent clients are simulated using local load generators (rather than real clients running at separate nodes from the server replicas). As we also observed, the throughput for Fast Paxos is actually lower in the presence of 3 or more concurrent clients. We believe that the lower throughput is due to collisions. As such, we conclude that Fast Paxos is best used for achieving lower end-to-end latency in the presence of a single client and it is not appropriate for use in high throughput systems.

Charron-Bost & Schiper (2006) described a scheme to minimize the cost of collision recovery by using an optimistic approach. The primary goal of this paper is drastically different from ours, which focuses on the development of an easy-to-implement recovery rules instead of optimizing the Fast Paxos even further.

There is an open source implementation of the Fast Paxos algorithm at http://libpaxos.sourceforge.net/paxosprojects.php#libfastpaxos. Unfortunately, from the reading of the source code, we could locate neither the code that handles conflict resolutions, and nor that for the value selection rule implementation.

OpenReplica is a well-known implementation of state-machine replication using Classic Paxos (Altinbuken and Sirer, 2012). However, our work is quite different from OpenReplica in two fronts: (1) We describe the theory and implement of Fast Paxos while OpenReplica only implements Classic Paxos; (2) The goal of our work is to thoroughly present the algorithmic-level details and intricacies of Fast Paxos as well as its use for state machine replication, while OpenReplica is designed to achieve high performance for practical systems. Similarly, the tutorial summary by Meling and Jehl (2013) presented the details of the Classic Paxos, while we focus on the issues unique to Fast Paxos, such as collision detection, value selection rule, and collision recovery.

CONCLUSION

In this article, we presented the theory and implementation of the Fast Paxos algorithm. The theory is described in an easier-to-understand way compared with the original article by Lamport (Lamport, 2006). In particular, we introduced a new approach to deriving the quorum requirements based on an intuitive value selection rule for the coordinator in cases of collisions without relying on the evaluation of O4(v), which is difficult to do in practice. As expected, our quorum requirements lead to exactly the same set of inequalities in (Lamport, 2006) for quorum formation based on the cardinality of the system.

We show that in a Fast Paxos implementation, a number of additional mechanisms are needed to cope with practical scenarios in a state-machine replication system. Furthermore, we conducted comprehensive experiments to evaluate the performance of Fast Paxos for state-machine replication. We show that Fast Paxos is most appropriate for use in a single client configuration. The presence of two or more concurrent clients even in a local area network would incur frequent collisions, which would reduce the system throughput and increase the mean response time as experienced by

clients. Due to frequent collisions, Fast Paxos actually performs worse than Classic Paxos in the presence of moderate to large number of concurrent clients.

ACKNOWLEDGMENT

I sincerely thank the reviewers for their constructive criticism and valuable suggestions on how to improve an earlier version of the article. This research is supported in part by an award from the Cleveland State University Graduate Faculty Travel Program.

REFERENCES

Altinbuken, D., & Sirer, E. G. (2012). *Commodifying replicated state machines with OpenReplica*. Retrieved from http://openreplica.org/static/papers/OpenReplica.pdf

Bolosky, W. J., Bradshaw, D., Haagens, R. B., Kusters, N. P., & Li, P. (2011, March). Paxos replicated state machines as the basis of a high-performance data store. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation* (pp. 11-11). USENIX Association.

Burrows, M. (2006, November). The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (pp. 335-350). USENIX Association.

Camargos, L., Madeira, E. R., & Pedone, F. (2006). Optimal and practical wab-based consensus algorithms. In Proceedings of Euro-Par 2006 Parallel Processing (pp. 549-558). Springer Berlin Heidelberg.

Camargos, L., Schmidt, R., & Pedone, F. (2008, July). Multicoordinated agreement protocols for higher availabilty. In *Proceedings of Network Computing and Applications*, (pp. 76-84). IEEE. doi:10.1109/NCA.2008.28

Chandra, T. D., Griesemer, R., & Redstone, J. (2007, August). Paxos made live: An engineering perspective. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing* (pp. 398-407). ACM. doi:10.1145/1281100.1281103

Charron-Bost, B., & Schiper, A. (2006, December). Improving fast paxos: Being optimistic with no overhead. In *Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing*, (pp. 287-295). IEEE. doi:10.1109/PRDC.2006.39

Hunt, P., Konar, M., Junqueira, F. P., & Reed, B. (2010, June). ZooKeeper: wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference* (*Vol. 8*, pp. 11-11). USENIX.

Junqueira, F., Mao, Y., & Marzullo, K. (2007). Classic paxos vs. fast paxos: Caveat emptor. In *Proceedings of USENIX Hot Topics in System Dependability* (HotDep). USENIX.

Lamport, L. (2001). Paxos made simple. *ACM Sigact News*, *32*(4), 18–25.

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

Lamport, L., & Massa, M. (2004, June). Cheap paxos. In *Proceedings of Dependable Systems and Networks*, (pp. 307-314). IEEE. doi:10.1109/DSN.2004.1311900

Mao, Y., Junqueira, F. P., & Marzullo, K. (2008, December). Mencius: Building efficient replicated state machines for WANs. In Proceedings of OSDI (Vol. 8, pp. 369-384). OSDI.

Meling, H., & Jehl, L. (2013). Tutorial summary: Paxos explained from scratch. In *Proceedings of OPODIS* (LNCS), (vol. 8304, pp. 1-10). Springer.

Rao, J., Shekita, E. J., & Tata, S. (2011). Using Paxos to build a scalable, consistent, and highly available datastore. *Proceedings of the VLDB Endowment*, *4*(4), 243–254. doi:10.14778/1938545.1938549

Vieira, G., & Buzato, L. E. (2008). On the coordinator's rule for Fast Paxos. *Information Processing Letters*, *107*(5), 183–187. doi:10.1016/j.ipl.2008.03.001

Vieira, G., & Buzato, L. E. (2013). *The performance of Paxos and Fast Paxos*. arXiv preprint arXiv:1308.1358.

Zhao, W. (2007, November). A lightweight fault tolerance framework for web services. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence* (pp. 542-548). IEEE Computer Society. doi:10.1109/WI.2007.18

Zhao, W. (2010). Building highly dependable wireless web services. *Journal of Electronic Commerce in Organizations*, *8*(4), 1–16. doi:10.4018/jeco.2010100101

Zhao, W. (2014). *Building dependable distributed systems*. John Wiley & Sons. doi:10.1002/9781118912744

Zhao, W., Melliar-Smith, P. M., & Moser, L. E. (2010, July). Fault tolerance middleware for cloud computing. In *Proceedings of the 3rd IEEE International Conference on Cloud Computing*, (pp. 67-74). IEEE.

Zhao, W., Zhang, H., & Chai, H. (2009). A lightweight fault tolerance framework for web services. *Web Intelligence and Agent Systems*, *7*(3), 255–268.

Zhao is currently an Associate Professor at the Department of Electrical and Computer Engineering, Cleveland State University. He earned his Ph.D. at University of California, Santa Barbara, under the supervision of Drs. Moser and Melliar-Smith, in 2002. Dr. Zhao has authored a research monograph titled: Building Dependable Distributed Systems *published by Scrivener Publishing. Furthermore, Dr. Zhao published over 80 peer-reviewed papers in the area of fault tolerant and dependable systems (three of them won the best paper award), computer vision and motion analysis, and material sciences. Dr. Zhao's research is supported in part by the US National Science Foundation, and by Cleveland State University. Dr. Zhao is currently serving on the technical program committee for numerous international conferences and is a member of editorial board for* International Journal of Performability Engineering, International Journal of Web Science*, and several international journals of the International Academy, Research, and Industry Association. Dr. Zhao is a senior member of IEEE and is currently serving on the executive board of the IEEE Cleveland Section.*