PAXOS: Consensus in a Distributed System

Departement INFORMATIK

Philipp Fath (philipp.fath@stud.unibas.ch)
04.11.2010

Supervisor: Dr. Diego Milano / Course: Distributed Information Systems
Abstract

“The Paxos algorithm, when presented in plain English, is very simple”

This is the whole abstract of Leslie Lamport’s paper “Paxos made simple”. Short and precise. I want however give it a try and make the ideas of consensus a bit more understandable. My aim is to introduce the reader basically to consensus and the Paxos algorithm.

Consensus recently gained immense importance due to the fact that modularity, distribution degree and independence level of systems and system parts are getting more and more popular. Achieving Consensus is a fundamental problem to be solved when implementing a fault-tolerant distributed system.

I will focus on simple consensus whose aim is to agree in the system on a single value among various proposed values in a distributed setting. Even in case of failures correctness in consensus must not be violated. So called non-Byzantine errors may arise at the different participants (“roles”) or at the messaging between these participants. Basic PAXOS algorithm is an implementation which guarantees consensus in such systems.
# Contents

Abstract ........................................................................................................................................ 2  
Contents ....................................................................................................................................... 3  
Introduction ................................................................................................................................ 4  
Leslie Lamport ............................................................................................................................... 4  
Consensus ...................................................................................................................................... 4  
Paxos Parliament ................................................................................................................................. 4  
Non-Byzantine Faults ...................................................................................................................... 5  
Upshot ............................................................................................................................................ 5  
Basic PAXOS ................................................................................................................................. 6  
Roles in a consensus model ............................................................................................................. 6  
Fault-tolerance ................................................................................................................................. 6  
Quorum ........................................................................................................................................... 7  
Basic PAXOS model ......................................................................................................................... 7  
Basic PAXOS algorithm ................................................................................................................... 8  
Progress ......................................................................................................................................... 9  
Conclusion ..................................................................................................................................... 10  
Examples ....................................................................................................................................... 10  
Types of PAXOS ............................................................................................................................ 10  
Sources ......................................................................................................................................... 12
Introduction

Leslie Lamport
This paper is mainly based on different articles published by Leslie Lamport. Leslie Lamport is an American mathematician, born in New York, February 1941. He studied mathematics at Massachusetts Institute of Technology (Cambridge/MA) and Brandeis University (Waltham/MA). After having received his Ph.D. he got famous as developer of LaTeX. One of his focuses in theoretical computer science is dedicated to distributed systems. He developed many solutions to fundamental problems in distributed systems. Among these, he invented the PAXOS-algorithm which can provide consensus in a distributed system, even in presence of certain faults. For his outstanding work Leslie Lamport received various prizes and awards. Just some examples are his five honorary doctorates or the Dijkstra prize in distributed computing. Presently he is working for Microsoft Research.

Consensus
Consensus originally comes from the Latin word “consentire” which could be translated as “feel together”. According to different definitions consensus could be understood as a general agreement in sense of unanimity among a group. When we talk about consensus we mean a judgment arrived among a group by most of the group. So consensus is group solidarity in sentiment and belief. Transferring these abstract definitions of consensus to distributed systems we want a system to have a consistent agreement on a value. As the system may be distributed we obligate it to agree consistently on that value in all of its components independent of their location. Especially there must not be given contradictory answers by different processes.

Paxos Parliament
To understand how significant consensus is to a distributed system and which faults may arise in such a system, Leslie Lamport introduces the ancient parliament at the Greek island of Paxos.

Imagine an ancient parliament located at the Aegean island of Paxos where trading of customs was the main business of the people living there. The parliament’s most important task was to determine the law. Law was determined by the sequence of decrees passed by the parliament. Instead of having a single secretary recording all the decrees passed in its sequence, every legislator maintained a ledger where he recorded the law. Even though this procedure seems to be very easy there result some difficulties with the flexibility of the legislators. Due to the fact that parliamentary duty was considered to be a voluntary avocation, legislators only worked part-time at parliament. Outside the parliament they all had much more important real jobs. Thus legislators continually wandered in and out the parliamentary chamber. Even during the process of passing a decree they could leave the chamber. Another problem was the bad acoustics in parliament. Legislators needed messengers to communicate with each
other. And also the messengers worked part-time in parliament. Hence messages might be delivered late or even never. However, all legislators and messengers were considered to act deeply honest. Especially nobody corrupted messages or lied.

Most important requirement in that parliament was of course to keep the ledgers of the legislators consistent in a way that they do not contain contradicting information. For example if there existed at any time a consensus in the chamber which decree was passed for decree number 140 in the sequence of the law, there must never exist a ledger having a different decree recorded at number 140. Of course there may exist ledgers which do not yet contain information on what decree was passed as number 140, but we do not want to have another decree fixed in any ledger at position 140.

This ancient parliament is an example for a distributed system which needs to ensure consensus within the whole system despite so called non-Byzantine faults in that system. The basic PAXOS algorithm is able to provide these requirements in such an environment.

Remark: This example is described by Leslie Lamport in his paper “The part-time parliament”, where he is claiming having derived the PAXOS algorithm from archeological discoveries. But the truth is that this ancient parliament never really existed at the island of Paxos. Leslie Lamport totally invented it to popularize his consensus algorithm named PAXOS.

Non-Byzantine Faults
Basic PAXOS algorithm assumes the presence of non-Byzantine faults. Non-Byzantine faults could be divided into three classes of faults:

- First, there are Fail-stops or Crashes. Processes in the system may work correct until a certain moment and then suddenly crash.
- Second, there might be so called Omission or Partial Discontinuation faults. Some messages might be lost during delivery, nevertheless they are all correct.
- Third, there can exist faults in Timing. This means that the execution of actions and the sending of messages may be early or late. Especially the order of message delivery might be different to the sending order.

Talking in our example about agents (legislators) and messages (via messengers) we can accept that agents operate at arbitrary speed, may fail by stopping or may restart. Messages are allowed to be delivered at arbitrary deliver time, can be duplicated or even be lost.

However not considered in basic PAXOS are Byzantine errors and inconsistency. Basic PAXOS cannot deal with incorrect working processes which send corrupted messages.

Upshot
We discovered that there exist distributed systems which underlie non-Byzantine faults. But anyway there is a need of consensus in these systems. PAXOS algorithm introduced by Leslie Lamport is able to build a fault-tolerant distributed system.
Basic PAXOS

Roles in a consensus model
In the model of the consensus algorithm we distinguish three different roles: proposers, acceptors and learners. These roles describe the behaviors of their particular classes of agents. There may exist several agents acting in the same role and it is possible that a process is performing different roles at the same time. Agents can communicate with each other by sending asynchronous messages.

Assuming a model with a single acceptor (Image 1) it would be quite easy to choose a value. If the acceptor accepts a value proposed by the proposer then the learner can learn that this value has been chosen in the system. To ensure that the value possibly ever be learned by a learner never changes (or respectively to preserve consensus in the system) we only need to obligate the acceptor to accept only one proposal.

![Image 1: Roles in consensus model](Image 1: Roles in consensus model)

Although this approach is quite easy it is very unsatisfactory. In case of the failure of the single acceptor information gets totally lost. If the acceptor is able to recover after a crash but not remembering a before accepted proposal, it may accept a new proposal and thus violate consensus.

Fault-tolerance
Due to the fact that agents and messages may be influenced by non-Byzantine faults we want to construct a more robust system. For that we introduce a set of acceptors (Image 2). Now a proposed value can be accepted by several acceptors. A proposal is chosen if a majority (at least a quorum [cf. next chapter]) of acceptor agents accepts it. If a learner notices that at least a quorum accepted the same proposal it learns the value of that proposal.

If acceptors are allowed to accept only one proposal the choice of the majority of acceptors leads to a non-ambiguous value to be learned, because it is impossible to find another majority of acceptors which voted on another proposal.
Quorum
A quorum is the term of the minimal quantity of acceptors which needs to stay non-faulty to ensure that a value may be learned. Normally this would be the smallest possible majority of the acceptors. To be chosen a proposal needs to be accepted by at least a quorum of the acceptors. Furthermore if there has been chosen a value before by any quorum of acceptors then there is in any other quorum at least one acceptor which was part of that “choosing” quorum. All different sets of acceptors composing a quorum have at least one acceptor in common. But a quorum needs not to be necessarily the majority of acceptors. We are allowed to put different weights on acceptors. In this case a quorum would consist of any set of acceptors holding together the majority of the weights. Using weighted acceptors makes sense if we consider some acceptors to be more reliable than others.

Basic PAXOS model
In the basic PAXOS model there is considered not only to be a set of acceptors but also various proposers and various learners (Image 3).
less at the same time. Assuming a set of three acceptors we can end up in a state in which every acceptor accepts a different value. Conclusion would be that there is no consensus (acceptors do not agree on a proposal) and system is blocked from further progress. To avoid this undesirable situation we have to allow the acceptors to accept several proposals. Nevertheless all safety requirements of consensus must be met at any time:

- Nontriviality: Only a value that has been proposed may be chosen
- Stability: Only a single value is chosen
- Consistency: Two learners cannot learn different values

Regarding the stability requirement it is important to distinguish between value and proposal. There may be chosen several proposals in the system, but they necessarily have the same value to not violate consensus. To discern different proposals every proposals has a unique proposal number $N$. Thus a proposal consists of a proposal number $N$ and a value $v$. Every proposer has a pool of unique numbers which it is using in an ascending order to number serially its proposals.

Basically PAXOS algorithm is a protocol which ensures in a 2-phase commitment that every new (higher-numbered) proposal does not violate consensus. Therefore it uses the local knowledge of the acceptors in a first prepare-phase and when having validated a value which is allowed to be proposed the algorithm is initiating a commit on this value.

**Basic PAXOS algorithm**

Basic PAXOS algorithm is executed in two phases. Every phase can be divided into two steps:

**Phase 1**

- A proposer selects out of its pool a unique proposal number $N$ and sends a prepare request with number $N$ to a majority (at least a quorum) of acceptors.
- When an acceptor receives a prepare request it responds with a promise not to accept any more proposals numbered less than $n$. If the acceptor already had answered to a higher-numbered prepare request before he will of course not answer (or send instead a non-acknowledge message [NACK]). The answer consists of the promise for number $N$ and enclosed to it the highest-numbered proposal (proposal number $N$ and value $v$) that it has accepted before (if any).

**Phase 2**

- If the proposer receives sufficient responses (from a majority of acceptors, at least a quorum) then it sends an accept request to each of those acceptors with number $N$ and value $v$. Value $v$ is the value of the highest-numbered proposal among the responses. If none of the acceptors accepted a proposal before it could be any value (most appropriately of course a value requested by a client).
- If an acceptor receives an accept request it accepts the proposal number $N$ with value $v$. Of course the proposal will not be accepted if the acceptor already answered a higher-numbered prepare request in meantime.
Progress
Besides having correctness in our consensus model it would be nice to ensure that progress is possible. In other words we are interested that a system can achieve a consensus and that the chosen value can be learned. Responsible for that are three elements.

- First, we have to obligate acceptors to accept proposals if this is not contradicted by prepare requests. If this rule would not exist there could be a quorum of acceptors never accepting any proposal and hence dead locking the system.

- Further we can easily imagine a situation of two (or more) dueling proposers which launch successfully prepare-requests with ascending proposal numbers in a round-robin sequence. Followed by these prepare-requests there would be send accept-requests which would fail due to the fact that another proposer already launched a higher-numbered prepare-request. Then consequently the proposer would decide to start a new higher-numbered prepare-request. Result would be that the other proposer cannot complete phase two successfully and may act in the same way. This scenario could be continued during infinite steps. Even though this problem does not affect correctness in the system it prevents the system from progressing and getting to a consensus. To guarantee that there do not exist dueling proposers there needs to be a leader among the proposers which is the only one allowed to send proposals. In what way that election is organized and how a by-election works in case of a crash of the leader is not important because it does not influence correctness in the system.

- Third, there needs to be assured that a large enough set of acceptors work non-faulty. At least a quorum of acceptors and a learner need to stay non-faulty over time to ensure liveness of the consensus. Not having a learner implies that no value can be learned. Not having a quorum of acceptors surviving means to lose at least a quorum and having left less than a quorum of acceptors. Losing a quorum means that there could be lost an already chosen proposal by that group. Furthermore can a system with less than a quorum of acceptors left never learn a chosen value. Progress will be impossible. Even worse is the situation if acceptors coming up again having an amnesia. Thus consensus could be violated. Preventing the system from losing its consensus and ensuring progress we require that there is at least a quorum of acceptors correct working over time.
Conclusion

Examples

- **The parliament at the island of Paxos**

  In the example of the ancient parliament, legislators would be agents of all three roles. Proposing a law a legislator would be a proposer agent. Accordingly to the PAXOS algorithm the proposing legislator needs first of all to send a prepare-request to the other legislators. All legislators act as acceptors at the same time. Receiving a prepare-request they send back their promises (if possible). Subsequently the proposing legislator will send his accept-message. Every legislator who finally accepted the decree sends to all other legislators (or alternatively to a leader) the message that he accepted that proposal. Receiving these messages the legislators act in the role of learner agents. Having received an accept notification of the majority of the legislators a legislator is able to fix the chosen decree into his ledger. Guaranteeing progress there should always be a majority of legislators present at the chamber and there should be a legislator acting as leading proposer.

- **Distributed Information Systems**

  A more newsworthy example where to use PAXOS algorithm is a system of distributed backup-databases. Several clients want to send a write command to a set of redundant databases. To maintain consistency the system needs to agree on the next command to be executed out of the proposed requests. The clients act as proposers, additionally we need a distributed set of acceptors. These could e.g. be located at the database-servers. Finally the databases act as learners.

**Types of PAXOS**

PAXOS is not a single algorithm but a whole family of algorithms.

- **Multi-PAXOS**

  Multi-PAXOS is the most common deployment of PAXOS. If there is the need of various instances of PAXOS (e.g. for a stream of agreed values) a lot of overhead is produced during phase 1. If the leader is relatively stable, all instances can use the same leader to reduce overhead. Phase 1 becomes unnecessary and may be left out in subsequent instances.

- **Cheap-PAXOS**

  Cheap PAXOS distinguishes between main processors and auxiliary processors. In a normal stable state only the main processors work, in unstable periods the auxiliary processors get activated and have to reconfigure the system. As the stable state should be the normal situation, auxiliary processors can be cheap and slow or normally devoted to other tasks.
• **Fast-PAXOS**

Fast-PAXOS saves message delays from a client’s request until its value gets learned. Basically there are introduced two optimization steps. First is the reduction of message delays by having clients which directly send requests to the acceptors. Second, when a recover technique is specified in advanced acceptors can handle collisions themselves uncoordinated and thus save message delays.

• **Generalized-PAXOS**

Aim of generalized PAXOS is not to agree on a single value but on an increasing set of values. Agreeing on values of a partially ordered set is important e.g. for command sequences in state-machines.

• **Byzantine-PAXOS**

Byzantine PAXOS is a version of PAXOS which is able to guarantee consensus in a system even in presence of Byzantine faults.
Sources

- **Papers**
  - Paxos Made Simple, Leslie Lamport (2001)

- **Online**