Consensus is a pervasive problem in many areas of human endeavor; consensus is the process of agreeing

to one of several alternatives proposed by a number of agents. We restrict our discussion to the case

of a distributed system when the agents are a set of processes expected to reach consensus on a single

proposed value.

No fault-tolerant consensus protocol can guarantee progress [123], but protocols that guarantee

freedom from inconsistencies (safety) have been developed. A family of protocols to reach consensus

based on a finite state machine approach is called Paxos.5

A fair number of contributions to the family of Paxos protocols are discussed in the literature. Leslie

Lamport has proposed several versions of the protocol, including Disk Paxos, Cheap Paxos, Fast Paxos,

Vertical Paxos, Stoppable Paxos, Byzantizing Paxos by Refinement, Generalized Consensus and Paxos,

and Leaderless Byzantine Paxos. He has also published a paper on the fictional part-time parliament in

Paxos [206] and a layman’s dissection of the protocol [207].

The consensus service consists of a set of n processes. Clients send requests to processes and propose a

value andwait for a response; the goal is to get the set of processes to reach consensus on a single proposed

value. The basic Paxos protocol is based on several assumptions about the processors and the network:

• The processes run on processors and communicate through a network; the processors and the network

may experience failures, but not Byzantine failures.6

• The processors: (i) operate at arbitrary speeds; (ii) have stable storage and may rejoin the protocol

after a failure; and (iii) can send messages to any other processor.

• The network: (i) may lose, reorder, or duplicate messages; (ii) sends messages are asynchronously

that may take arbitrarily long times to reach the destination.

The basic Paxos considers several types of entities: (a) client, an agent that issues a request and waits

for a response; (b) proposer, an agent with the mission to advocate a request from a client, convince

the acceptors to agree on the value proposed by a client, and act as a coordinator to move the protocol

forward in case of conflicts; (c) acceptor, an agent acting as the fault-tolerant “memory” of the protocol;

(d) learner, an agent acting as the replication factor of then protocol and taking action once a request

has been agreed upon; and finally, (e) the leader, a distinguished proposer.

A quorum is a subset of all acceptors. A proposal has a proposal number pn and contains a value v.

Several types of requests flow through the system: prepare, accept.

In a typical deployment of an algorithm, an entity plays three roles: proposer, acceptor, and learner.

Then the flow of messages can be described as follows [207]: “Clients send messages to a leader; during

normal operations the leader receives the client’s command, assigns it a newcommand number i , and then

begins the i -th instance of the consensus algorithm by sending messages to a set of acceptor processes.”

By merging the roles, the protocol “collapses” into an efficient client/master/replica-style protocol.

A proposal consists of a pair, a unique proposal number and a proposed value, (pn, v); multiple

proposals may propose the same value v. A value is chosen if a simple majority of acceptors have

accepted it.We need to guarantee that at most one value can be chosen; otherwise there is no consensus.

The two phases of the algorithm are described here.

Phase I

1. Proposal preparation: A proposer (the leader) sends a proposal (pn = k, v). The proposer

chooses a proposal number pn = k and sends a prepare message to a majority of acceptors

requesting:

a. that a proposal with pn < k should not be accepted;

b. the pn < k of the highest number proposal already accepted by each acceptor.

2. Proposal promise: An acceptor must remember the highest proposal number it has ever

accepted as well as the highest proposal number it has ever responded to. The acceptor can

accept a proposal with pn = k if and only if it has not responded to a prepare request with

pn > k; if it has already replied to a prepare request for a proposal with pn > k, then it

should not reply. Lost messages are treated as an acceptor that chooses not to respond.

Phase II

1. Accept request: If the majority of acceptors respond, the proposer chooses the value v of the

proposal as follows:

a. the value v of the highest proposal number selected from all the responses;

b. an arbitrary value if no proposal was issued by any of the proposers.

The proposer sends an accept request message to a quorum of acceptors including (pn = k, v).

2. Accept: If an acceptor receives an accept message for a proposal with the proposal number

pn = k, itmust accept it if and only if it has not already promised to consider proposals with a

pn > k. If it accepts the proposal, it should register the value v and send an accept message to

the proposer and to every learner; if it does not accept the proposal, it should ignore the request.

The following properties of the algorithm are important to showits correctness: (1)Aproposal number

is unique; (2) any two sets of acceptors have at least one acceptor in common; and (3) the value sent out

in Phase II of the algorithm is the value of the highest numbered proposal of all the responses in Phase I.

Figure 2.15 illustrates the flowofmessages for the consensus protocol.Adetailed analysis of themessage

flows for different failure scenarios and of the properties of the protocol can be found in [207].We

onlymention that the protocol defines three safety properties: (1) nontriviality, the only values that can be

learned are proposed values; (2) consistency, at most one value can be learned; and (3) liveness, if a value v has been proposed, eventually every learner will learn some value, provided that sufficient processors

remain non-faulty. Figure 2.16 shows the message exchange when there are three actors involved.

In Section 4.5 we present a consensus service, the ZooKeeper, based on the Paxos protocol. In

Section 8.7 we discuss Chubby, a locking service based on the algorithm.