Blog

Written words that are often helpful

Paxos Algorithm, in English

Posted by chathaway on Jan. 14, 2020, 7:20 a.m.

The Paxos Algorithm

The Paxos algorithm is used to handle decision making in a decentralized way, which is tolerant of faultsand other such problems. It is usually applied to network programming and distributed systems.

(For background, see the Background section at the end of this post.)

High-level

The thing that tripped me up most was understanding how to apply this algorithm to an actual problem.

Part of the difficulty is that it is usually thought about as a single 'round' in a larger machination; that is, whenpeople describe the Paxos algorithm, they describe the algorithm that is used to accept a single value.This does not help with a long-living daemon-like process, since we expect the entire round to end in a relatively shorttimeframe (under most cases, but it is not actually bounded and could go on forever; see the 'Leadership' section formore details).

When talking about a single round of the Paxos algorithm, what we're aiming to do is have a proposal sent out toa group of acceptors, who will accept it and, once concensus is reached, informed the 'learners' about the new value.The learners are the ones who then pass the data onto interested parties.

Itemize what is just said, we have 3 roles:

1. The proposers
2. The acceptors
3. The learners

Some diagrams also include a 'client', who initiates a request to the proposer, but I omit it here since I don't consider it important to understanding the core algorithm.

It is not uncommon to have a single process do all 3 of these things, but for the protocol to work, we need to thinkabout them seperately.

There are 2 stages of communication between the proposers and acceptors, which can go on indefinitely.After those 2 stages are done, there is a single stage for the learners.

First, the proposer sends out a message to the acceptors asking them to promise not to listen to anything that happenedearlier."Earlier" here is usually relative, and simply uses an incrementing integer to indicate a serialized sequence.

The proposer sends this message to at least N/2 + 1 acceptors (where N is the total number of acceptors), and waits forconfirmation responses. This first message is usually called 'prepare'.

Each acceptor will do one of two things:

1. Note down the time of the message, and reply 'yes'
2. Important note; if this acceptor made it to stage 2 before and accepted a value, they must include what that value was in the response. The goal is to make sure that a consensus is reached, even if it's not the one that the proposer is hoping for.
3. This message is usually called 'promise'
4. Observe that the number is less than or equal to the previous number they saw, and ignore the message
5. Some people suggest being nice and letting the proposer know that you won't accept the proposal.

After the proposer gets enough votes, it moves to the next stage.

If the proposal timed out (because there was no response from some number of acceptors, or there was a kind responsesaying that the proposal was not accepted), it begins again, incrementing the integer used to serialize the sequence.

In the next stage, the proposer sends a message to all acceptors asking them to accept a value.The value the proposer is asking them to accept is either the value they want, or, the value returned to them asa response from one of the acceptors. If the acceptors returned a value, it must be that value.

Unless the acceptors have made a promise between the time of making a promise to this proposer and when they get themessage, they will accept the response and send it to the learners. If they have made a new promise, they simply ignore the message.

Pseudo Code

First, an implementation for the proposers.We assume there is a driver who constructs the Proposer and invoked handle_promise anytime we get a new message.

```
class Proposer:  def __init__(self, acceptors, value):    self.proposal_number = 0    self.acceptors = acceptors    self.value
```

The acceptor might look something like this:

```pythonclass Acceptor: def **init**(self, learners): self.learners = learners self.latest_promise = 0 self.value = None

def handle_prepare(self, proposer): if proposer.proposal_number <= self.latest_promise: proposer.send(negative()) self.latest_promise = proposer.proposal_number proposer.send(promise(self.latest_promise, self.value))

def handle_accept(self, proposer): if proposer.proposal_number <= self.latest_promise: return self.value = proposer.value for learner in self.learners: learner.send(result(self.value))```

The learner should do nothing more than verify that it gets the same answer from all acceptors.A closed feedback loop between the learner and the proposer may allow it to try and send a new value again, if theproposal it was hoping to get through didn't win, but then we start getting into the more generic database work on ACIDsystems.

## Leadership

A clever observer may note that it is possible to get stuck in a loop of extracting promises from the acceptors, butnever actually delivering a value.It is very unlikely that this loop would go on forever (due to things like physics, the halting problem etc.), butit would slow down an implementation.The normal way to work around this is to elect a leader, who will be the only person making proposals.

In my opinion, this is a practice of impossibilities.The odds of ending up in a situation in which no consensus can bereached could be better handled by introducing a random (small) delay before sending a new proposal after each NACK fromthe acceptors.

## Background

Paxos is an algorithm proposed by Lamport in his 1998 paper (originally envisioned earlier, but this was the only sourceI could find).The goal is to have a group of nodes come to consensus such that a majority agree on a particular value (or, in theoriginal paper, decree).It is useful in scenarios where messages may go missing, get delayed for arbitrarily long times, etc.

A good way of thinking about it is this:

1. A senate convenes once a year to make a decree, but not all members of the senate are available in person.
2. To make the decree, those in chamber send out a warning saying they are going to send version 1 of the decree, and to ignore all earlier version.
3. Once a majority of the members (remote or not) agree, they send out the decree.
4. Once the decree is accepted by a majority of the members (remote or not) and the sent decree receives confirmation, it is official.

After the decree is sent out, the senate meets again next year.

This roughly approximates the iterative approach of Paxos; we agree to a single proposal, then begin the next.We could, for example, send a sequence of messages electing a node as 'the leader':1. All nodes send out a decree saying "I am the leader"2. The node which has its message accepted by a majority of acceptors is the leader.

The only thing we know for sure is that after each message/decree is accepted, it is official.But, we can construct application logic around this to do something *useful* with the Paxos algorithm, like electa leader (for example, if someone else declared themselves to be the leader in the next round, we could choose toignore the decree based on deterministic rules about response time; but we would still acknowledgethe decree).

---



---
```