Investigating papers about Paxos algorithm

Antonio Paolacci

June 16, 2011

1 Byzantine Fault Tolerance

Byzantine fault tolerance is a specific field studying about fault tolerance mechanisms. The object of Byzantine fault tolerance is to be able to defend agaist byzantine failures. The byzantine term in a PC-distributed enviroment means "malicious behaviour" of a system node. A byzantine node could fail in a "arbitrary ways", not just crashing and stopping activities, but also sending bad messages or bad requests, flooding the system and corrupting his internal state. All the algorithm to provide a service in a byzantine fault tolerace manner are based on the assumption that there are not too many Byzantine faulty nodes and a majority set(**Quorum**) of correct processes run with safety the algorithm. Fault tolerance is insured via "state machine replication" based on principle of rathen than running a single istance of a server, running multiple istances. All istances should receive the same client request in the same order and be deterministic.

2 Paxos protocol

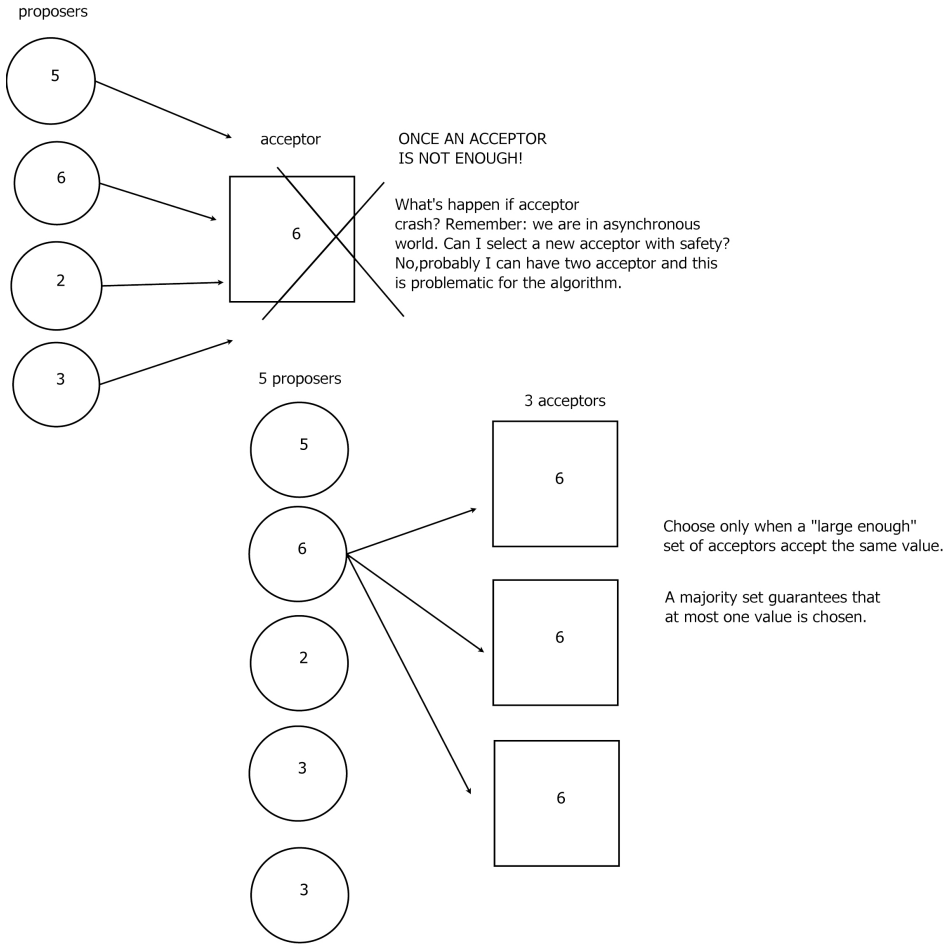
Paxos is a consensus protocol on distributed system model. Most simple model's assumption is: a set of process that can propose integer values; processes can crash and recover, processes can have access to stable storage; asynchronous communication via messages; messages can be lost,duplicated but not corrupted. The goal of paxos protocol is:

- only a value that has been proposal can be chosen,
- only a single value is chosen,
- a process never learns that a value has been chosen unless it has been.

Two types of proprerties:

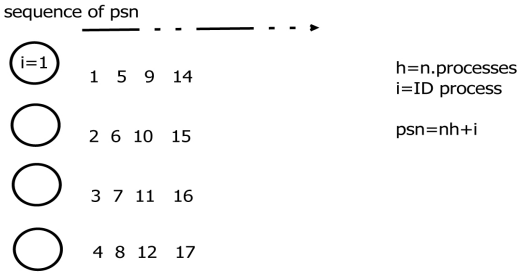
- **safety** don't do things that are not allowed,
- **liveness** do something useful:
  - some proposed value is eventually chosen,
  - if a value is chosen a process eventually learns it.

Three distinct roles of paxos nodes:*proposer*,*acceptor*,*learner*. Acceptor: Any message sent to an acceptor must be sent to a quorum of acceptors. Proposer: a proposer invokes a *client* request, attempting to convince the acceptors to agree on it. Learner: Once a client request has been agreed on by the acceptors, the learner may take action(execute the request and send a response to the client for example).
How can we choose a single value?



Property P1 An acceptors must accept the first proposal that it receives. Consequence of an agreement on the same value is that an acceptor

must accept more than one proposal value. To keep track of different proposal, assign a natural number to each proposal. A proposal is then a pair $< psn, value >$. Different proposals have different psn(proposal sequence number) based on a determinist rule:



A proposal is chosen when it has been accepted by a majority of acceptors. A value is chosen when a single proposal with that value has been chosen.

Property P2 If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v.

Property P2a If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v.

P2a and P1 $\Rightarrow$ problem!

Property P2b If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v.

How would we prove that P2b holds? We would assume that some proposal with psn m and value v is chosen and show that any proposal issued with psn $n > m$ also has value v. By induction on n: every proposal in m...(n-1) has value v. For the proposal number m to be chosen, there must be some set C consisting of a majority of acceptors such taha every acceptors of C accepted it.
P2b is satisfied in m...(n-1), wich implies: every acceptors in C has accepted a proposal with number in m...(n-1) and every proposal with number in m...(n-1) accepted by any acceptor has value v.
Since any set S containing a majority of acceptors contains at least one member of C, we can conclude that a proposal numbered n has value v by ensuring that the following invariant is mantained:

Property P2c For any v and n, if a proposal with value v and psn n is issued, then there is a set S consisting of a majority of acceptor such that:

- (a) no acceptor in S has accepted any proposal numbered less than n
- (b) v is the value of highest-numbered proposal any all proposal numbered less than n accepted by the acceptor in S

How we ensure P2c? A proposer chooss a new proposal number n and send a request to each member of some set of acceptors asking it to respond with:

- (a) a promise never again accept a proposal numbered less than n
- (b) the proposal with the highest number less than n that it has accepted, if any.

⇒ **prepare request**

If the proposer receives the requested responses (from a majority of acceptors), then it can issue a proposal with number n and value v,where v is the value of the highest-numbered proposal among the responses, or is any value selected by the proposer if the responder reported no proposal.

⇒ **accepted request**

What about acceptor?

- An acceptor can ignore any request,
- an acceptor can always reply to a prepare request,
- an acceptor can reply to an accept request if and only if: it has not promised not to.
- **Property P1a** An acceptor can accept a proposal number n iff it has not responded to a prepare request having a number greater than n.

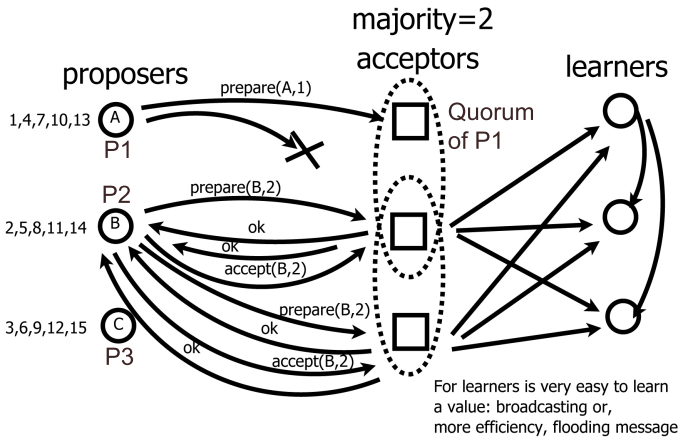
2.1 Summary

Phase 1:

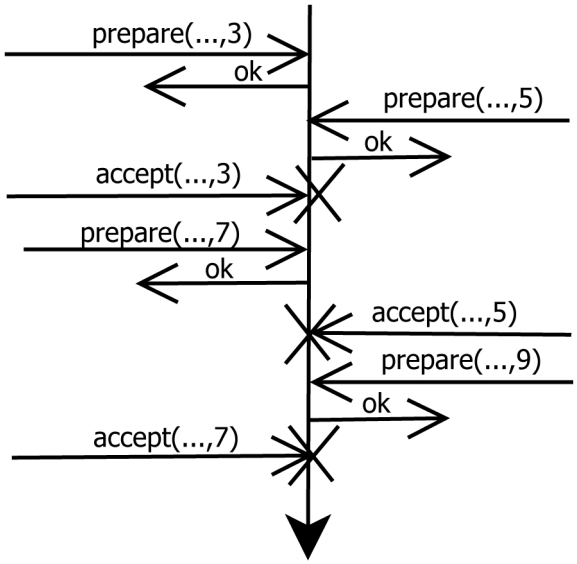
- (a) a proposer send a prepare request,
- (b) any acceptors replies to the prepare request.

Phase 2:

- (a) if the proposer receive a majority of replies it send an accept request,
- (b) an acceptor accepts an accept request unless it responded to a prepare request with higher sequence number.



What about liveness in paxos algorithm? An acceptor could never accepts a request!



2.2 Remark about liveness of Paxos

Liveness is only ensured when there is a single proposer in the system. This fact can be ensured differentiating proposers by electing a leader with a

leader election distributed algorithm. Remember that we are in asynchronous enviroment and probably multiple proposers believe themselves to be leaders. For instance the current leader may fail and later recover, but the other proposers have already re-elected a new leader. The recovered leader has not learned this yet and attempts to begin a round in conflict with the current leader.

2.3 Practical Paxos

How do we use Paxos to implement a replicated state machine? We start for each stable storage a new session of Paxos protocol with proposer,acceptor and learner nodes.

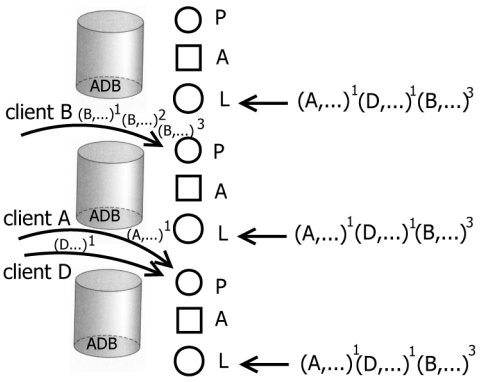


Figure 1: The figure shows as the i-th client request is accepted and performed by Paxos subsystem.

3 Paxos in malicious world: Byzantizing Paxos

It is possible to derive a Byzantine Paxos algorithm from a distributed non-Byzantine one by a procedure called "Byzantizing", which converts an N process algorithm that tolerates the benign failure of up to f processes into an N + f process algorithm that tolerates f Byzantine processes. In the Byzantized algorithm, the N good processes emulate the execution of the original algorithm despite the presence of f Byzantine ones. In Paxos we assume N acceptors and if we want to tolerate f fault, a quorum of acceptors has N-f acceptors. For Safety we require that have a non-empty intersection, which is true if $N > 2f$. An acceptor can vote for at most one value in any ballot. A value v is chosen in a ballot iff a quorum of acceptors have voted

for v in that ballot. A value is chosen iff it is chosen in some ballot. We say that a value v is safe at a ballot number b if no value other than v has been chosen or ever can be chosen in any ballot numbered less than b. (Although described intuitively in temporal terms, *safe at* is actually a function of the algorithm's current state.) The algorithm maintains the following properties:

- P1. An acceptor can vote for a value v in ballot b only if v is safe at b.
- P2. Different acceptors cannot vote for different values in the same ballot.

Phase 1a The ballot-b leader sends a 1a message to the acceptors.

Phase 1b An acceptor responds to the leader's ballot-b 1a message with a 1b message containing the number of the highest-numbered ballot in which it has voted and the value it voted for in that ballot, or saying that it has cast no votes.

Phase 2a Using the 1b messages sent by a quorum of acceptors, the leader chooses a value v that is safe at b and sends a 2a message containing v to the acceptors.

Phase 2b Upon receipt of the leader's ballot-b 2a message, an acceptor votes for v in ballot b by sending a 2b message.

- P3a. If no acceptor in the quorum has voted in a ballot numbered less than b, then all values are safe at b.
- P3b. If some acceptor in the quorum has voted, let c be the highest-numbered ballot less than b in which such a vote was cast. The value voted for in ballot c is safe at b. (By P2, there is only one such value.)

In Byzapaxos we have N acceptors and f faulty nodes that can behave arbitrarly.
There's a way for the destination process of a message to know what is the sending process, based on public and private key, authentication code, cryptographic tools.
We say that we have real and fake acceptors. We define the set of byzantine acceptors to be the union of the set of real and fake acceptors. We define *byzaquorum* to be the set of *byzacceptors* that is guaranted containing a quorum (of real) acceptors. If a quorum of acceptors contains q acceptors then a byzaquorum contains q+f acceptors.
What does it means?

Paxos: a learner learns a value when it receives a message 2b from a quorum of acceptors.
ByzPaxos: a learner learns a value when it receives a message 2b from a byzaquorum of acceptors.
The key action in Paxos consensus is the leader's Phase 2a action, which chooses a safe value based on properties P3a and P3b.
The leader can deduce that P3a holds if it receives 1b messages from a byzquorum, each asserting that the sender has not voted, because that byzquorum contains a quorum of acceptors. However, P3b is problematic. In the original algorithm, it is satisfied if there is a 1b message from some single acceptor reporting a vote in a ballot c. However, in the Byzantized algorithm, there is no way to determine if a single message is from a real or fake acceptor. To mantain safety we requires that a vote be reported in the highest numbered ballot c by f+1 byzacceptors. $N = 2f + 1, N - 2f > f, N > 3f, N = 3f + 1$

- P3a'. If there is no ballot numbered less than b in which f+1 acceptors have voted, then all values are safe at b.
- P3b'. If there is some ballot c in which acceptors have voted and there is no higher-numbered ballot less than b in which f +1 acceptors have voted, then the value v voted for in c is safe at b.

The Phase 2a action is then always enabled by the receipt of 1b messages from a byzquorum because, if P3a0 does not hold, then we can apply P3b0 with c the largest ballot in which f +1 byzacceptors have voted for the same value. However, this is unsatisfactory because it leads to a Byzantine consensus algorithm requiring more than 4f acceptors.
$\Rightarrow N > 4f, N = 4f + 1$

References

[1] L. Lamport. The Part-Time Parliament. ACM Transactions on Computer Systems, vol. 16, n. 2, pp.133-169, 1998.

[2] L. Lamport. Paxos Made Simple. ACM SIGACT News vol. 32, n. 4, pp. 51-58, 2001.

[3] Vivien Quéma, Lectures Distributed System Seminars, 2011