# Distributed Systems

William Schultz

December 19, 2022

## Fault Tolerance

There are some fundamental requirements to establish bounds for fault tolerance in an omission fault model. If an arbitrary set of $f$ nodes can fail by stopping at any time, then this means that if we want a protocol that makes progress, we would need to ensure that any "work" we do (e.g. executing operations, writing down data, etc.) is made sufficiently redundant so that it can be accessed even in the case of maximum node failure. So, this implies we need to write all data to at least $f+1$ nodes, so that there is always at least one non-faulty node with the data we need to access. So, this seems to imply that having simply $f+1$ total nodes might be sufficient for a fault tolerant protocol. But, this still doesn't satisfy a progress requirement, since if we now need to write everything down to $f+1$ nodes, then failure of $f$ out of $f+1$ nodes clearly stalls our protocol, since it can do any work safely. So, our additional requirement is that both

1) we must write any work down to $f+1$ nodes and

2) that we always have $f+1$ non-faulty nodes available that we can write work down on

Thus, this naturally gives us a total node requirement of

$$n = f + (f+1) = 2f + 1$$

That is, even in the case of $f$ maximum node failures, we will always have $f+1$ nodes available to us to write down our work, allowing us to make progress.

## PBFT

### Model

The work on Practical Byzantine Fault Tolerance [1] considers an asynchronous distributed system where nodes are connected by a network which can fail to deliver messages, delay them, or deliver them out of order. Furthermore, it considers a Byzantine failure model i.e., faulty nodes may behave arbitrarily, subject only to the above restrictions. They do assume, however, cryptographic techniques that prevent spoofing and can detect corrupted messages. In other words, Byzantine processes can send any message, but we assume the identity of the sender of a message can be determined by the receiver [2]. This can be achieved this with public-key signatures [3], message authentication codes (MACs), etc.

### Intuitions and Algorithm

If we assume a starting point of a classic 2-phase Paxos consensus approach, the following are some of the essential issues that arise and must be dealt with when we add in Byzantine faults:

1. **Leader equivocation**: if a leader is faulty (Byzantine), then it can trivially send two conflicting messages in the same view (i.e. with the same proposal number). This means that, for example, it could send out and accept message with its own proposal number but with a different value to each replica. Then, we would end up with a quorum of replicas having accepted that proposal, but they all have different values, so which one is the true value to agree upon?

2. **Wrong value adoption**: A leader (faulty or not) that accepts a wrong value (i.e. not highest among previously) chosen can lead to safety violation as considered in the standard 2-phase Paxos model.

A key idea of the algorithm is about how we deal with the issue of potentially Byzantine leaders. That is, we need to protect against leaders sending conflicting messages to different followers such that we would violate the constraints needed to ensure safety in, for example, classic asynchronous consensus Paxos in the standard omission (non Byzantine) fault model. If a leader is faulty and just went ahead and followed the standard 2-phase protocol used in Paxos (*prepare* + *accept*), then in the *prepare* phase it could tell different replicas arbitrarily different things i.e. tell them to accept one value and then change this value

Why exactly is it bad if a leader is Byzantine i.e. even in classic Paxos?

The essence of the algorithm is as follows:

1. Primary sends a PRE-PREPARE$(value, p)$ message for view/proposal number $p$.
2. Replica responds to the first PRE-PREPARE message it receives from a primary.
3. Primary gathers PRE-PREPARE responses from $n-f$ replicas, and then sends PREPARE$(v, proof)$ (note this message may be linear in size since it contains signed codes fro m up to $n$ nodes.)
4. If a replica sees PREPARE$(value, p, proof)$ and $proof$ contains $n-f$ valid signatures for PRE-PREPARE$(value, p)$, then it goes ahead and accepts.
5. Primary then gathers $n-f$ PREPARE responses from replicas.

Note that since we assume a public key infrastructure (PKI) set up between nodes of the system, any node can securely verify that a message was signed by some another node.

## Notes

- Given $n = 3f + 1$ nodes, for any 2 quorums with $n - f = 2f + 1$ nodes, we are guaranteed they intersect in at least $f + 1$ nodes (just draw a picture). Note that if you talk to at least $f + 1$ nodes then you are sure you are in contact with at least one non-faulty (non-Byzantine) node.

# References

[1] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186, USA, 1999. USENIX Association.

[2] Leslie Lamport. Byzantizing paxos by refinement. In *Proceedings of the 25th International Conference on Distributed Computing*, DISC'11, page 211–224, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, feb 1978.