Radoslaw Krzeski
Summary of Topic VIII - BFT

Wikipedia article – Byzantine Fault Tolerance
(
The article states that it is the "dependability of a fault-tolerant computer system". In distributed computing systems, components may fail and may be imperfect information on whether a component has failed. The name is derived from the Byzantine Generals Problem. The system may have difficulties reaching a consensus with other as to whether deem it failed and shut it out of the system. Other words for it are interactive consistency, source congruency and error avalanche.

A Byzantine fault is essentially failure where other parties or connected system fails to reach a consensus on whether the individual element has failed. The tolerance is the ability to withstand such problems, and where an agreement is needed to solve the problem. A correctly functioning system will be able to continue operating even with faulty elements.

BFT is considered one of most the difficult classes of failure modes. It implies no restriction as to the fault, where fail-stop simply assumes faulty nodes. Byzantine failures can even assume working but "corrupt" nodes, and confuse other systems further. Can arise from human or electrical fault.

The ground problem, derived from the paper introduced in 1982 by Lamport, Shostak and Pease, describe a problem around generals each commanding an army. They give orders, but when many of them are involved, a consensus problem emerges, and the ninth general may choose to split his message selectively. Messengers introduce another difficulty sphere. It could also be cast into "null", triggering predefined, backup solutions. This story then places with computers in them as actors. Other issues are bounded delay (needs synchronous delivery, with the example of Commander A, and Lietutant B and C), and unforgeable message signatures, but where a single bit change may crash the system.
Several systems have been developed to tackle this issue.

Practical Byzantine Fault Tolerance System was built in 1999 to test the consensus problem on a larger scale (thousands of requests, millisecond increase in latency). UpRight and BFT SmaRT-Library are similar systems.

The Byzantine Fault Tolerance in bitcoin is solved through the proof-of-work methodology and to reach a coherent global view of the system state.

Practical Byzantine Fault Tolerance – article by Castro & Liskov
In this paper, they called it a replication algorithm that can tolerate Byzantine faults. Created for the future to tackle errors and attacks on nodes. It is designed to work with the internet and asynchronous environments., with improved response time.

Attacks are increasing in numbers as our systems become more complicated and attractive to hack. This can cause nodes to fail and BFT tolerance a crucial component of any system. The Practical BFT provides a better way for state machine replication, and offers liveness and safety. Where n-1/3 nodes are faulty.
Earlier models were too theoretical or to slow to run efficiently or even rely on known bounds. Previous systems also have the risk of being the victim of DoS-attack. The Practical BFT is not vulnerable to this, since it can work asynchronous, and uses crypto keys when faults come in play. It supports the NFS protocol and does not assume synchronicity.

In essence it is the first system to correctly survive Byzantine fault in asynchronous networks, describes a number of optimization for use in real systems, provides a resistant file system and quantifies a cost for a replication system.

The system model does however assume a synchronous system. Nodes may fail. They run different systems, different administrator. Operating systems can be bought, different programmers provide different implementations.
Cyptotechniques are used to prevent spoofing and replays to detect corrupt data. It also assumes a very strong adversary that can coordinate attacks, delay coms and correct nodes.

The algorithm(service properties) can implement any deterministic replicated service with a state and some operations. It provides both safety and liveness assuming once again the N-1/3 faulty nodes.
3f + 1 is the minimum number if replicas that allow an asynchronous system to provide the safety and liveness when up to f replicas are faulty.
The algorithm is a state machine replication that is replicated on different nodes in the distributed systems. Each state machine maintains service state and service operations. They are also called replicas, which move through configurations called views. In a view one replica is the primary and the others are backups. Views, and when moved through consecutively, are numbered consecutively View changes are carried out when it seems that the primary has failed.

1.A client sends a request to invoke a service operation to the primary
2.The primary multicasts the request to the backups

3.Replicas execute the request and send a reply to the client
4.The client waits for f+1 replies from different replicas with the same result; this is the result of the operation.

Replicas have two requirements; must be deterministic and they must all start in the same state. Safety ensured through the agreement of a total order for the execution of requests. A client sends requests and uses timestamps. It uses views to determine the current primary that broadcast messages to all backups.
Client waits for f+ 1 replies with valid signatures. If no reply received quick enough it is then broadcasted to replicas, and a system comes in place where both replicas and primary verify each other requests. This process also goes through the phases called pre-prepare, prepare and commit.

View changes are triggered by timeouts that prevent backups from waiting indefinably for requests to execute. Most requests and messages are also sent using message authentication codes(MACs). They can compute three orders faster than digital signatures.

From this point on I jumped to the conclusion. I found the paper highly technical in the way it describes the BFT system they implemented in the PBFT, and difficult to summarize easily. Essentially the system tolerates greater node faults and asynchronous systems. Optimizations such as message authentications codes improves performance replacing public-key signatures.

The Quest for Scalable Blockchain Fabric

A paper looking at the Proof-of-work vs BFT Replication. It argues the traditional way bitcoin is built is not scalable, and currencies such as Ethereum needs far better solutions to support its arbitrary distributed applications. This new way, requires the transitions to classical state-machine replication and BFT variants of systems.

Distributed consensus wasn't proven effective until Bitcoin came along and came close with its probabilistic agreement. Transactions are all ordered on the blockchain. Two magical numbers control some of the difficulty controlling the mining of the coin, block frequency and block size. With about an hour time, 6 new blocks usually approve transactions, or at least it being the recommended time to confirm. The system eventually became slow.

A smart contract can be modeled as a state machine. Consistent execution across multiple nodes can be achieved using state replication. A member of these system is BFT, promising consensus even with faulty nodes. Three decades later, it can support thousands of transactions per second, but struggles to scale in the amount of nodes applicable in a system.

PoW based systems offer good node scalability but poor performance, while BFT good performance for a small number of replicas, and not being very-well explored.

Important consideration to take into aspects when comparing PoW and BFT;
-node identity management(permissionless/permissioned) vs/ (logically centralized identity management),
-consensus finality/dually (block are final)(forks),
-scalability (mining pool risks),
-performance,
-power of adversary (hashing power vs node faults),
-network synchrony,
-existence of correctness proofs; underlying blockchain mechanics.

The following gives nice overview of the some of the differences:

	PoW consensus	BFT consensus
Node identity management	**open, entirely decentralized**	permissioned, nodes need to know IDs of all other nodes
Consensus finality	no	**yes**
Scalability (no. of nodes)	**excellent (thousands of nodes)**	limited, not well explored (tested only up to $n \leq 20$ nodes)
Scalability (no. of clients)	**excellent (thousands of clients)**	**excellent (thousands of clients)**
Performance (throughput)	limited (due to possible of chain forks)	**excellent (tens of thousands tx/sec)**
Performance (latency)	high latency (due to multi-block confirmations)	**excellent (matches network latency)**
Power consumption	very poor (PoW wastes energy)	**good**
Tolerated power of an adversary	$\leq 25\%$ computing power	$\leq 33\%$ voting power
Network synchrony assumptions	physical clock timestamps (e.g., for block validity)	**none for consensus safety** (synchrony needed for liveness)
Correctness proofs	no	**yes**

BFT have never initially been tested beyond 10 to 20 nodes, due to intensive network coms.

Further, some improvements have been proposed, such the GHOST protocol(Greedy Heaviest- Observed Sub-Tree), resolved conflict by weighing subtrees rooted in blocks rather than longest chain rooted in blocks.
Another is Bitcoin-NG, which uses a leader-based block that accumulates microblock before another leader block is chosen. BlockDAG proposes parallel forks, with non-conflicting transaction being on parallel chains, before being merged in a later block. BFT communities have also researched parallelization heavily.
Other improvements have tried to decrease the overhead over information needed in BFT systems (XFT) or even merge PoW and BFT protocols. Moving parts of the cryptographic techniques to hardware-bases solutions also seems to be a promising solution.

References:

https://en.wikipedia.org/wiki/Byzantine_fault_tolerance

http://pmg.csail.mit.edu/papers/osdi99.pdf

https://allquantor.at/blockchainbib/pdf/vukolic2015quest.pdf