XFT: PRACTICAL FAULT TOLERANCE BEYOND CRASHES
ABOUT FAILURE MODEL…

- Byzantine Failure
- Crash Failure
- No Failure

Synchronous vs. Asynchronous

- (Authenticated) BFT: f+1
- XPaxos: 2f+1
- PBFT: 3f+1
- CONSENSUS: f+1
- Paxos: 2f+1
3F+1 IS TOO MUCH...

- In practice, 3f+1 is too much
- PBFT assumes the adversaries could cause any number of network partition: too strong!
- What if we could quantify the “degree of asynchronous” of the network into f?
A replica is partitioned if it is not in the largest subset where everyone can communicate within $\Delta$

Non-partitioned replicas forms a synchronous (sub)network
At time $s$, three disjoint sets of node failures:

- $t_p(s)$: # of correct but partitioned replicas
- $t_c(s)$: # of crash replicas
- $t_{nc}(s)$: # of byzantine replicas
The System is non-anarchy if:

- \( t_c(s) + t_p(s) + \lfloor (n-1)/2 \rfloor \leq (n-1)/2 \), at any time \( s \)
  - The rest is the “good” (synchronous and correct) part of the network
  - This model permits \( t_{nc}(s) = \lfloor (n-1)/2 \rfloor \), as long as \( t_c(s) + t_p(s) = 0 \!\)
  - The roles of failures could be switched at anytime

Cross Fault-Tolerant (XFT): a protocol is XFT if it maintains correctness in any time the system is non-anarchy
USE CASES OF XFT

- Accidental non-crash fault
- Geo-replicated replicas
- Permissioned blockchain
Feature 1: For normal operation, the quorum (Synchronous group $sg_i$) for view $i$ is pre-determined
- In each $sg_i$: 1 leader and $f$ followers (active replica)
- The rest: passive replica; learn values lazily
- At least one correct synchronous replica in each $sg_i$
- $sg_i$ is supposed to be in the “good” part of network

Feature 2: decentralized view-change: after detecting $sg_i$ is ”bad”
Upon receive REQ, the primary $S_i$ does:
1. Assign slot $sn$ to req
2. Signs prep=<PREPARE, $D(req)$, $sn$, $i$>$\sigma_{si}$
3. Logs <req, prep> into PrepareLog[$sn$]
4. Send <req, prep> to all active replica

Upon receive <req, prep> $S_j$ does:
1. Update $sn$
2. Logs <req, prep> into PrepareLog[$sn$]
3. Signs commit=<COMMIT, $D(req)$, $sn$, $i$>$\sigma_{sj}$
4. Send commit to all active replica

Upon receive $f$ matching signed COMMIT $S_j$ does:
1. Logs COMMIT into CommitLog[$sn$]
2. Execute it
3. Send REPLY to client

Client wait for $f+1$ matching REPLY from all active replicas

Then commit
VIEW CHANGE

- Any replica $s_j$ in $sg_i$ initial view change if one of the following happens
- (1) Receives misbehaved message
- (2) Retransmission timer of $s_j$ expires
- (3) $S_j$ does not complete view change toward $i$ timely
- (4) Receive a SUSPECT

- A decentralized approach: every active replica in $sg_{i+1}$ retrieves info committed in $sg_i$
To start a view change, sj:
1. Stops activity in sgi
2. Sends <SUSPECT, i, sj>_{σ_j} to all other replicas

Received SUSPECT from active replica in view i, sj does:
1. Stops activity in sgi
2. Sends <VIEW-CHANGE, i+1, sj, CommitLog>_{σ_j} to all active replica in sgi+1

Received at least t+1 VIEW-CHANGE after Waiting for at least 2 Δ, sj of sgi+1 does:
1. Inserts the VIEW-CHANGES in VCSET
2. Sends <VIEW-FINAL, i+1, sj, VCSET>_{σ_j} to all active replica in sgi+1

Received VIEW-FINAL from all replica in sgi+1, sj of sgi+1 does:
1. Taking the union of the sets as VCSET*
2. For each slot num sn select the commit log with highest view number in VIEW-CHANGE as committed
3. New primary sends PrepareLog to followers
- If a client c has already committed $req$ in view $i$ with slot $sn$, and if in any view $i'>i$, some correct replica $s_k$ commit $req'$ for $sn$

$\Rightarrow req = req'$

- Proof: if $req$ committed by $c$ in view $i$

$\Rightarrow$ recv. $f+1$ matching $REPLY$ in $i$

$\Rightarrow (1)$ every correct replica in $sg_i$ log $req$ in $CommitLog[sn]$
In view $i+1$, every correct replica in $sg_{i+1}$ waits for $f+1$ VIEW-CHANGE.

Then replicas in $sg_{i+1}$ exchange these info through VC-FINALs.

Outside anarchy, at least one “good” $s_j$ in $sg_{i+1}$

=> (2) the correct replica $s_k$ that commits message $req'$ at $sn$ in view $i+1$ must has received a VC-FINAL from $s_j$

$s_j$ waits for $f+1$ VIEW-CHANGE

=> (3) must contains information from a “good” replica from $sg_i$
- By (1)(2)(3) we can see how the correct information transfer from $s_{g_i}$ to $s_{g_{i+1}}$:
- From a “good” replica in $s_{g_i}$
- $\rightarrow$ a “good” replica in $s_{g_{i+1}}$ (by VIEW-CHANGE)
- $\rightarrow$ all correct replica in $s_{g_{i+1}}$ (by VC-FINAL)
PERFORMANCE EVALUATION

- Benchmark: Zyzzyva, PBFT, Paxos
- Amazon EC2 worldwide cloud platform
FAULT-FREE PERFORMANCE

- Compared with PBFT: simpler message
- Compared with Zyzzyva: less stress on leader
- Performance close to Paxos
CPU usage of XPaxos is higher than others, due to the use of digital signatures.
PERFORMANCE UNDER FAULTS

- View Change is less than 10 sec
MACRO-BENCHMARK ON ZOOKEEPER

- XPaxos and Paxos outperform the others
- XPaxos beats Zab: $t$ followers v.s. $2t$ followers
XPaxos tradeoffs the "degree of asynchronous" for less replicas.

All members in a synchronous group are supposed to be in the “good” part of the network to remain functional, which could be varying in time and may take a long time to be found.

High communication complexity in decentralized view change.
THANK YOU!!!