XFT: Practical Fault Tolerance Beyond Crashes – Liu et al.

Presented by Ryan Feng
How do we (practically) move beyond crash faults?

• BFT – expensive and cautious

• Is there something between Paxos (CFT) and PBFT (BFT)?
Motivation

• What if we could...
  • Perform comparably to Paxos with just crash faults?
  • Make stronger reliability guarantees than Paxos?
  • Tolerate some Byzantine faults under a realistic threat model?

• ... all with only 2f + 1 replicas?

Enter XFT!
Intuition

• BFT

• XFT

Key Insight: Allowing the Byzantine node to arbitrarily control the network is a stronger model than is often necessary. Separate the two!
Terminology

- Correct: no faults
- Benign: correct or crash failure
- Non-crash: non-crash Byzantine faults
- Partitioned: not in largest synchronous group
- Anarchy: lack a majority of correct, synchronous replicas

Modified Fig. 1, Liu et al. 2016
XFT Goal: Stay Safe in the Absence of Anarchy
When is this useful?

- Aka, when are worst case conditions like anarchy unlikely?

- “Accidental” Crash Faults
- Wide-Area Networks (WAN) and geo-replicated systems
- Blockchain
Fault model comparison

<table>
<thead>
<tr>
<th>Model</th>
<th>Consistency</th>
<th>Non-Crash Faults</th>
<th>Crash Faults</th>
<th>Partitioned Replicas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous CFT (e.g., Paxos [28])</td>
<td>0</td>
<td>n</td>
<td>n - 1</td>
<td>n - 1</td>
</tr>
<tr>
<td></td>
<td>availability</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Asynchronous BFT (e.g., PBFT [9])</td>
<td>[n-1/3]</td>
<td>n</td>
<td>n - 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>availability</td>
<td></td>
<td></td>
<td>[n-1/3] (combined)</td>
</tr>
<tr>
<td>(Authenticated) Synchronous BFT (e.g., [29])</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XFT (e.g., XPaxos)</td>
<td>0</td>
<td>n</td>
<td>n - 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>availability</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: The maximum numbers of each type of fault tolerated by representative SMR protocols. Note that XFT provides consistency in two modes, depending on the occurrence of non-crash faults.

Liu et al. 2016
XPaxos

• 3 Components
  • Common-case Protocol
  • Decentralized View Change
  • Fault Detection (FD) mechanism

• Algorithm summary:
  • Carve out an $f + 1$ (or $t + 1$) replica synchronous group as the *active* replicas
  • On view change, all new active replicas retrieve state from old active replicas
  • Detect issues by comparing logs
Common Case (t >= 2)

- General case
  - Client sends signed REQ
  - Primary assigns sequence number, logs / sends signed PREPARE
  - (Active) Followers accept if it’s the next seq. number and logs it in PREPARE log
  - Followers log / send COMMIT’s, wait to receive COMMITS from all active replicas, check they match PREPAREs, execute request

Liu et al. 2016
Common Case \((t = 1)\)

- **Special Case \((t = 1)\)**
  - Client sends signed REQ
  - Primary assigns sequence number, logs PREPARE / COMMIT, sends COMMIT
  - Follower accepts if it’s the next seq. number, logs it in COMMIT log, executes, sends commit with reply digest to Primary
  - Primary receives COMMIT from follower, executes if it matches PREPARE, compares results from execution, and responds to client with Follower’s digest

Liu et al. 2016
View Change

- Configurations chosen and known a priori
- Enumerate all possible groups of active replicas and rotate through
  - Note: need to transfer state and its proof to new replicas on view changes
- Leader is uniquely determined in each view

<table>
<thead>
<tr>
<th>Active replicas</th>
<th>Primary</th>
<th>$s_0$</th>
<th>$s_0$</th>
<th>$s_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Follower</td>
<td>$s_1$</td>
<td>$s_2$</td>
<td>$s_2$</td>
<td></td>
</tr>
<tr>
<td>Passive replica</td>
<td>$s_2$</td>
<td>$s_1$</td>
<td>$s_0$</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Synchronous group combinations ($t = 1$).
View Change Mechanics

- Initiated if:
  - Message is invalid
  - Retransmission timer expires
  - Prior view change is too slow
  - Receives SUSPECT message

- Processing
  - Stop processing, broadcast SUSPECT
  - Send VIEW-CHANGE to new active replicas with COMMIT logs
  - Compile VIEW-CHANGE info and send VC-FINAL across new active replicas
  - New Primary sends NEW-VIEW with PREPARE log info

Figure 3: Illustration of XPaxos view change: the synchronous group is changed from \((s_0,s_1)\) to \((s_0,s_2)\).
Fault Detection

• Goal: Identify machines suffering non-crash faults that could cause inconsistency if anarchy were to occur

• Approach:
  • Modify view change to additionally send prepare logs
  • Sketch of reasoning focuses on data loss faults
  • A correct follower would be able to use its commit log to detect a prior primary omitting an entry from both prepare and commit logs
Optimizations

• Checkpointing
  • Speeds up view changes and allows garbage collection, done every CHK requests

• Lazy Replication
  • Speed up view change state transfer
  • Each active follower lazily propagates commit log to one passive replica

• Batching and Pipelining
  • Saves time with crypto signing
  • Batch together up to B requests at a time before sending off prepare
Correctness Arguments (High-Level)

• Consistency
  • A benign client’s request for a given sequence number must match all benign replica’s commits for all later views
  • Client must have gotten response from a correct + synch. replica in previous view, which would have shared that info via the VIEW-CHANGE and VCFINAL process
  • More details in paper

• Availability
  • With every combo of active replicas, eventually there will be t + 1 correct, synchronous replicas
Experimental Setup

• Use Amazon EC2 instances across world
• Geo-replicated WAN settings
• Compare with:
  • WAN optimized Paxos
  • PBFT
  • Zyzzyva (speculative BFT)
• Test:
  • Without faults
  • With faults
  • With ZooKeeper
Fault-Free Results

Liu et al. 2016
Fault Performance Results

Figure 6: XPaxos under faults.

Liu et al. 2016
Real-world Benchmark: ZooKeeper

Figure 7: Latency vs. throughput for the ZooKeeper application ($t = 1$).

Liu et al. 2016
Reliability (High-Level)

• Calculate odds of consistency based on odds of faults, counting the possible configurations

• XPaxos vs. CFT:
  • XPaxos is strictly more consistent
  • Number of 9’s added increases with the more reliable a network is

• XPaxos vs. BFT:
  • Depends on network, odds of a correct process
  • Special case of $t = 1$: XPaxos stronger if $p_{\text{correct}} \times p_{\text{synchrony}} > p_{\text{benign}}^{1.5}$
Conclusion

• Take advantage of the pessimistic nature of general BFT, guarantee consistency in the absence of worst-case anarchy settings

• Provides an “in-between” protocol that is comparable to Paxos in CFT but can also tolerate some arbitrary Byzantine faults

• Offers a faster, more practical protocol to allow for some Byzantine faults, especially in geo-replicated settings