XFT: Practical Fault Tolerance Beyond Crashes

By Luke Brandl
Despite years of research, BFT systems have not yet been widely adopted in practice, largely due to:

- Increased resource use (e.g., number of replicas needed)
- Increased throughput overhead
- Unnecessarily high guarantees due to a very strong adversary
A Worthy Adversary

This stems from the Byzantine assumption that the adversary is quite powerful and can control:

- The faulty machines (memory errors malicious code, etc)
- The message delivery system across the entire network

The authors of XFT believe this is much stronger than what must be prevented in practice
CROSS FAULT TOLERANCE (XFT)

- Attempts to solve most practical BFT cases with minimal resource and performance overhead
- Uses State Machine Replication (SMR), similar to EVE
- Claims solid results
XFT

- Uses no more replicas compared with asynchronous (eventually synchronous) CFT, $2t+1$
- Provides all reliability guarantees of asynchronous CFT in the absence of Byzantine faults
- Guarantees safety and liveness even when Byzantine faults do occur when not in anarchy
- In some cases can offer strictly stronger reliability guarantees than state-of-the-art BFT
- NOTE: This is not the full strength of the classical BFT adversary
XFT Basics

XFT describes reliable protocols that tolerate crash machine faults regardless of the number of network faults and additionally tolerate non-crash machine faults when the number of machines that are either faulty or partitioned is within a threshold.
XFT Conditions

Under certain conditions XFT performs very well, in particular when network faults are limited. Examples include:

- Systems that are not susceptible to malicious attacks where network faults (asynchronicity) can be considered largely independent of stray ‘accidental’ non-crash faults
- Wide-area networks and geo-replicated systems
When those conditions are not met...

ANARCHY

- A severe system condition that causes liveness and even safety guarantees to fail
- There is no correct and synchronous majority of replicas, and at least one non-crash fault
- XFT assumes this case is extremely infrequent
XPaxos, an implementation

XPaxos is designed for good performance in geo-replicated settings, and is composed of three key components:

- Common case protocol for replicating and ordering requests across replicas.
- View change protocol to allow decentralized information transfer.
- Fault detection, to prevent non-crash faults from accumulating.
Luckily, anarchy is often not an issue. During normal operation, XPaxos works as expected.

- Leader sends req and the sequence number to active replicas
- Active followers send commit
- When $t$ commits received, execute req and send reply to client

Figure 2: XPaxos common-case message patterns (a) for the general case when $t \geq 2$ and (b) for the special case of $t = 1$. The synchronous groups are $(s_0,s_1,s_2)$ and $(s_0,s_1)$, respectively.
COMMON CASE ($t=1$)

When $t=1$, the process is even simpler, as no PREPARE is needed.

But what happens when anarchy appears?

Figure 2: XPaxos common-case message patterns (a) for the general case when $t \geq 2$ and (b) for the special case of $t = 1$. The synchronous groups are $(s_0, s_1, s_2)$ and $(s_0, s_1)$, respectively.

http://orig04.deviantart.net/6e10/f/2012/022/8/f/anarchy_punk_head____by_the_real_toto-d4n9ari.jpg
Synchronous Groups

Say there are $n=2t+1$ replicas total

- XPaxos denotes $t+1$ replicas as members of a synchronous group
- Each view number $i$ determines the synchronous group
- A view consists of one primary replicas, $t$ follower replicas, and the remainder as passive replicas.
- Change view when a machine or network fault within the synchronous group
VIEW CHANGES

● When view changes, all members of the new synchronous group try to transfer the state from the preceding view to the new one
● This guarantees that some correct replica from the new synchronous group will be able to contact some correct replica from any old synchronous group
● Choosing the new view is not particularly important

<table>
<thead>
<tr>
<th>Synchronous Groups $(i \in \mathbb{N}_0)$</th>
<th>$s_{gi}$</th>
<th>$s_{gi+1}$</th>
<th>$s_{gi+2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active replicas</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary</td>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>Follower</td>
<td>$s_1$</td>
<td>$s_2$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>Passive replica</td>
<td>$s_2$</td>
<td>$s_1$</td>
<td>$s_0$</td>
</tr>
</tbody>
</table>

Table 2: Synchronous group combinations $(t = 1)$. 
WHEN TO CHANGE VIEW

For some active replica $s_j$

- $s_j$ receives a message from another active replica that does not conform to the protocol (e.g., an invalid MAC signature)
- The retransmission timer at $s_j$ expires
- $s_j$ does not complete a view change to view $i$ in a timely manner
- $s_j$ receives a valid SUSPECT message for view $i$ from another replica in $s_{gi}$
How to Change View

- Each replica sends a signed message with `<VIEW-CHANGE, i+1, its id, Commit Log>` to each of the replicas in $s_{j+1}$
- If the replica is active in the new view, after waiting $2\Delta$ to receive (hopefully) $n-t$ VIEW-CHANGE messages, it sends a message with `<VC-FINAL, i+1, its id, VCSET>` where VCSET contains all of the view change messages to each replica active in the new view.
- The new primary then sends message `<NEW-VIEW, i+1, Prepare Log>`, where Prepare Log is the prepare logs generated in view $i+1$ for each selected request
Fault Detection

If XPaxos enters anarchy nearly all of its guarantees fail.

To remedy this, it uses fault detection to guarantee:

‘If a machine $p$ suffers a non-crash fault outside anarchy in a way that would cause inconsistency in anarchy, then XPaxos detects $p$ as faulty (outside anarchy)’

In order to achieve this, processes exchange both their commit logs as well as their prepare logs.
Fault Detection Example

Say $t=1$, so there is only a primary and a follower in the SG. In order for consistency to risk being violated, the primary needs to lose data in both its commit and prepare log.

The correct follower would reply in the view change with an entry in the primary’s prepare log causally precedes the respective entry in the follower’s commit log. The case for a faulty follower is similar.
Optimizations

- **Checkpointing** to speed up view changes

- **Lazy Replication** where each active replica lazily sends the commit log to one passive replica

- **Batching and Pipelining** where the primary batches several requests
Fault Free Performance

Confirming the low overhead of XPaxos in a fault free system. The results are organized by two request sizes (1kB, 4kB), one reply size (0kB) and $t$

Note in particular that XPaxos is very similar to simple CFT Paxos
Performance under Faults

- Crashes at 180, 300, and 420 seconds
- View change completed within 10 seconds
- Crashes last 20 seconds
IN SUMMARY

- XFT is a reliability improvement on traditional CFT, without the performance and resource (replica) overhead of full asynchronous BFT systems.
- In environments where significant and frequent communication failures are not likely, it performs reasonably well.
- XFT is an example of slightly loosening up constraints leading to huge performance and resource benefits.
- Avoid paying in performance for levels of security you don’t need.
NEW DIRECTIONS AND QUESTIONS

- How does XFT compare other BFT systems for higher \( t \)?
- In what other cases can constraints be loosened to improve performance, with minimal resource hit?
- How does XFT perform under various faults of different types, and more complex scenarios (e.g. \( t=45 \; n = 100 \))
- Combining XFT with hybrid fault models (VGT)

Questions?