ZYZZYVA: SPECULATIVE BYZANTINE FAULT TOLERANCE

Presenter: Buting Ma
**Motivation**

- To improve the efficiency of PBFT
- "Get rid of" Commit Phase
- Other techniques

<table>
<thead>
<tr>
<th></th>
<th>PBFT</th>
<th>Q/U</th>
<th>HQ</th>
<th>Zyzzyva</th>
<th>State Machine Repl. Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost</strong></td>
<td>3f+1</td>
<td>5f+1</td>
<td>3f+1</td>
<td>3f+1</td>
<td>3f+1 [31]</td>
</tr>
<tr>
<td>Replicas with application state</td>
<td>2f+1 [41]</td>
<td>5f+1</td>
<td>3f+1</td>
<td>2f+1</td>
<td>2f+1</td>
</tr>
<tr>
<td>Throughput</td>
<td>2+(8f+1)/b</td>
<td>2+8f</td>
<td>4+4f</td>
<td>2+3f/b</td>
<td>2†</td>
</tr>
<tr>
<td>Latency</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2/3‡</td>
</tr>
<tr>
<td>Critical path NW 1-way latencies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Properties of state-of-the-art and optimal Byzantine fault tolerant service replication systems tolerating $f$ faults, using MACs for authentication [9], and using a batch size of $b$ [9]. Bold entries denote protocols that match known lower bounds or those with the lowest known cost. †It is not clear that this trivial lower bound is achievable. ‡The distributed systems literature typically considers 3 one-way latencies to be the lower bound for agreement on client requests [11, 21, 25]; 2 one-way latencies is achievable if no concurrency is assumed. We include detailed discussion of this table in our extended technical report [17].
Basic Ideas

- Deferring
  - Similar to EVE
  - Fix divergence when it actually happens
- Consistency check at client
  - Similar to voter
- Move work from agreement to view-change
  - One less phase (commit) in agreement sub-protocol
  - One more phase (I-hate-primary) in view-change sub-protocol
AGREEMENT

- if the client receives $3f + 1$ mutually-consistent responses (including an application-level reply and the history on which it depends), then the client considers the request complete and acts on it.

- if the client receives between $2f + 1$ and $3f$ mutually-consistent responses, then the client gathers $2f + 1$ responses and distributes this commit certificate to the replicas. Once $2f + 1$ replicas acknowledge receiving a commit certificate, the client considers the request complete and acts on the corresponding reply.

Figure 1: Protocol communication pattern within a view for (a) gracious execution and (b) faulty replica cases. The numbers refer to the main steps of the protocol numbered in the text.
AGREEMENT DETAILS

1. Client sends request to the primary.
2. Primary receives request, assigns sequence number, and forwards ordered request to replicas.
3. Replica receives ordered request, speculatively executes it, and responds to the client.

4a. Client receives $3f + 1$ matching responses and completes the request.
AGREEMENT DETAILS

4b. Client receives between $2f + 1$ and $3f$ matching responses, assembles a commit certificate, and transmits the commit certificate to the replicas.

4b.1. Replica receives a commit message from a client containing a commit certificate and acknowledges with a local-commit message.

4b.2. Client receives a local-commit messages from $2f + 1$ replicas and completes the request.
AGREEMENT DETAIL

4c. Client receives fewer than $2f + 1$ matching spec-response messages and resends its request to all replicas (get around primary), which forward the request to the primary in order to ensure the request is assigned a sequence number and eventually executed.

4d. Client receives responses indicating inconsistent ordering by the primary and sends a proof of misbehavior to the replicas, which initiate a view change to oust the faulty primary. (spec-response contains local history of replicas)
Log and checkpoint

- Replicas can not globally confirm its own spec-response others (because no commit phase), until a check point/ view-change.
- A tentative checkpoint is considered stable when collect f+1 matching ones.
**View-change difference**

First, to ensure liveness, Zyzzyva strengthens the condition under which a correct replica commits to a view change by adding a new “I hate the primary” phase to the view change sub-protocol.

A replica in this phase will NOT be silent (keep processing), until it collects f+1 “I hate the primary” message (make sure the view-change will definitely happen), and then commits to view-change and becomes silent.
**View-change difference**

Second, to guarantee safety, Zyzzyva weakens the condition under which a request appears in the history included in the new-view message.

We no longer require a commit certificate but also allow a sufficient number of order-req messages to support a request’s ordering. This change ensures that the protocol continues to honor ordering commitments for any request that completes when a client gathers $3f + 1$ matching speculative responses.
**View-change Detail**

VC1. Replica initiates the view change by sending an accusation against the primary to all replicas.

VC2. Replica receives $f + 1$ accusations that the primary is faulty and commits to the view change.

VC3. Replica (new primary) receives $2f + 1$ view change messages.

VC4. Replica receives a valid new-view message and sends a view confirmation message to all other replicas.

VC5. Replica receives $2f + 1$ matching view confirm messages and begins accepting requests in the new view.
OPTIMIZATIONS

Shift the costs needed to deal with a faulty primary from the critical path (the agreement protocol) to the view-change sub-protocol, which is run only when the primary is faulty.

Implementing:
- Replacing Signatures with MACs.
- Separating Agreement from Execution.
- Request Batching.
- Caching Out of Order Requests.
- Read-Only Optimization.
- Single Execution Response.
- Preferred Quorums.

# of messages: $O(n^2)$ to $O(n)$

Less network congestion
**Performance**

Figure 3: Realized throughput for the 0/0 benchmark as the number of client varies for systems configured to tolerate $f = 1$ faults.

Figure 4: Latency for 0/0, 0/4, and 4/0 benchmarks for systems configured to tolerate $f = 1$ faults.

Figure 5: Latency vs. throughput for systems configured to tolerate $f = 1$ faults.
Figure 6: Fault scalability: Peak throughputs
Figure 7: Fault scalability using analytical model
Figure 8: Realized throughput for the 0/0 benchmark as the number of client varies when $f$ non-primary replicas fail to respond to requests.
• “Zyzzyva” is an interesting name.

• The paper explains in an intuitive way, easy to understand.

• Is the client too responsible? E.g. Malicious client causes liveness problem.

• Is it efficient enough to used in real applications?