Zyyzyva: Speculative Byzantine Fault Tolerance

Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong

Presented by Madelyn Gatchel
EECS 591 Fall 2021
## Byzantine Fault Tolerance (BFT) Replication

### System Model

<table>
<thead>
<tr>
<th>System Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Asynchronous system</td>
</tr>
<tr>
<td>• Unreliable channels</td>
</tr>
</tbody>
</table>

### Crypto

<table>
<thead>
<tr>
<th>Crypto</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Public/private key pairs</td>
</tr>
<tr>
<td>• Signatures</td>
</tr>
<tr>
<td>• Collision-resistant hashes</td>
</tr>
</tbody>
</table>

### Service

<table>
<thead>
<tr>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Byzantine clients</td>
</tr>
<tr>
<td>• Up to $f$ Byzantine servers</td>
</tr>
<tr>
<td>• $n &gt; 3f$ total servers</td>
</tr>
</tbody>
</table>

### System Goals

<table>
<thead>
<tr>
<th>System Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Always safe</td>
</tr>
<tr>
<td>• Live during periods of synchrony</td>
</tr>
</tbody>
</table>

[Adapted from EECS 591 Lecture 12 Slide 20]
PBFT Review
PBFT Review

- Client sends message to primary
PBFT Review

Client
Primary
Replica 1
Replica 2
Replica 3

• Client sends message to primary

Three-phase commit:
1. Pre-prepare
PBFT Review

• Client sends message to primary

Three-phase commit:
1. Pre-prepare
2. Prepare
PBFT Review

- Client sends message to primary

Three-phase commit:
1. Pre-prepare
2. Prepare
3. Commit
Client

Primary

Replica 1

Replica 2

Replica 3

1 2 3

• Client sends message to primary

Three-phase commit:
1. Pre-prepare
2. Prepare
3. Commit

• Replicas execute and send reply to client
The three-phase commit protocol is expensive.
Introducing...Zyzzyva

- **Novel contribution:** replicas speculatively execute requests without 3-phase commit
- Correct replicas may be inconsistent
- Replicas may send different responses to clients
- Clients use history and replies to detect inconsistencies
- Clients wait until history and speculative reply are stable to complete request

[Image of a red weevil]

**Zyzzyva:** tropical weevil and last word in dictionary

[Link to USA Today article](https://www.usatoday.com/story/news/nation-now/2017/06/27/zyzzyva-newest-last-word-oxford-english-dictionary-explained/431203001/)
Why Zyzzyva?

• State-of-the-art BFT protocols
  • Practical Byzantine Fault Tolerance (PBFT) [Castro and Liskov, 1999]
  • Query/Update (Q/U) [Abd-El-Malek et al., 2005]
  • Hybrid-Quorum replication (HQ) [Cowling et al., 2006]

• HQ replication paper => best technique depends on workload
• How does Zyzzyva solve this issue?
### BFT State-of-the-Art Comparison

**Gray/bold = best**

<table>
<thead>
<tr>
<th></th>
<th>PBFT</th>
<th>O/U</th>
<th>HQ</th>
<th>Zyzzyva</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total replicas w/ app state</td>
<td>3f+1</td>
<td>5f+1</td>
<td>3f+1</td>
<td>3f+1</td>
</tr>
<tr>
<td></td>
<td>2f+1</td>
<td>5f+1</td>
<td>3f+1</td>
<td>2f+1</td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC ops at bottleneck server</td>
<td>2+(8f+1)/b</td>
<td>2+8f</td>
<td>4+4f</td>
<td>2+3f/b</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Critical path NW 1-way latencies</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

[Adapted from Table 1]
Zyzzyva Overview

• One primary, 3f replicas
• Execution proceeds as a sequence of views
• Design challenges
  • Conditions for client request completion
  • Defining subprotocols to ensure correctness
• Subprotocols:
  - Agreement Orders requests for replica execution
  - Checkpoint Limits state replicas must store and reduces cost of view changes
  - View Change Coordinates new primary election if current is faulty or system is running slowly
Node State & Checkpoint Subprotocol
Node State & Checkpoint Subprotocol
Agreement Subprotocol
1. Client sends request to the primary.

\[ \langle \text{REQUEST}, o, t, c \rangle_{\sigma_c} \]
Agreement Subprotocol

1. Client
2. Primary receives request
   • Assigns sequence number
   • Forwards ordered request to replicas
Agreement Subprotocol

\[ \text{OR} = \langle \text{ORDER} - \text{REQ}, v, n, h_n, d, ND \rangle_{\sigma_p} \]

\[ \langle \langle \text{SPEC} - \text{RESPONSE}, v, n, h_n, H(r), c, t \rangle_{\sigma_i}, i, r, \text{OR} \rangle \]

1. Replica receives ordered request
2. Speculatively executes request
3. Responds to the client
Agreement Subprotocol

4. Client gathers speculative responses
Client Completion Summary

If client receives...

Exactly
\(3f + 1\) speculative response messages
Agreement Subprotocol

4a. If client receives exactly $3f + 1$ matching responses:
• Client completes the request.
Client Completion Summary

If client receives...

- **Exactly** $3f + 1$ speculative response messages
  - Complete request

- **Between** $2f + 1$ and $3f$ matching responses
4b. If client receives between $2f + 1$ and $3f$ matching responses:

- Client assembles a C-certificate
- Transmits it to the replicas
4b.1.
- Replica receives a COMMIT message from a client containing a C-certificate
- Replica acknowledges with a LOCAL-COMMIT message.
4b.2. If client receives a *LOCAL-COMMIT* message from $2f + 1$ replicas:

• Client completes the request.
Client Completion Summary

If client receives...

- **Exactly $3f + 1$ speculative response messages**
  - Complete request
  - Replica sends $LOCAL-COMMIT$ message

- **Between $2f + 1$ and $3f$ matching responses**
  - Make C-Certificate
  - Send $COMMIT$ to replicas

- **Fewer than $2f + 1$ matching responses**
  - If $2f + 1$ $LOCAL-COMMIT$ messages from replicas, complete request

Send $COMMIT$ to replicas
4c. If client receives fewer than $2f + 1$ matching responses:
4c. If client receives fewer than $2f + 1$ matching responses:

- Client resends its request to all replicas
Agreement Subprotocol

\[ m = \langle \text{REQUEST}, o, t, c \rangle_{\sigma_c} \]
\[ \langle \text{CONFIRM} - \text{REQ}, v, m, i \rangle_{\sigma_i} \]

4c. If client receives fewer than \(2f + 1\) matching responses:
- Client resends its request to all replicas
- Replicas forward the request to the primary
Client Completion Summary

If client receives...

- **Exactly** $3f + 1$ speculative response messages
  - Complete request
  - Replica sends $LOCAL-COMMIT$ message

- **Between** $2f + 1$ and $3f$ matching responses
  - Make C-Certificate
  - Send $COMMIT$ to replicas
  - If $2f + 1$ $LOCAL-COMMIT$ messages from replicas, complete request

- Fewer than $2f + 1$ matching responses
  - Retransmit request to all replicas
  - All replicas forward to primary so request is eventually executed

- **Inconsistent ordering** of messages
4d. If client receives responses indicating inconsistent ordering by the primary:

- Client sends a proof of misbehavior to the replicas.
- Replicas initiate a view change to oust the faulty primary.
4d. If client receives responses indicating inconsistent ordering by the primary:

- Client sends a proof of misbehavior to the replicas
4d. If client receives responses indicating inconsistent ordering by the primary:
- Client sends a proof of misbehavior to the replicas
- Replicas initiate a view change to oust the faulty primary.
Client Completion Summary

If client receives...

- Exactly $3f + 1$ speculative response messages:
  - Complete request
  - Replica sends LOCAL-COMMIT message

- Between $2f + 1$ and $3f$ matching responses:
  - Make C-Certificate
  - Send COMMIT to replicas
  - If $2f + 1$ LOCAL-COMMIT messages from replicas, complete request

- Fewer than $2f + 1$ matching responses:
  - Retransmit request to all replicas
  - All replicas forward to primary so request is eventually executed

- Inconsistent ordering of messages:
  - Send proof of misbehavior message
  - Initiate view change
View Change Subprotocol
View Change Subprotocol

• To ensure liveness
  • Extra “I hate the primary” phase added
  • A correct replica will not abandon a view unless every other correct replica does as well

• To guarantee safety
  • Weakens condition under which a request appears in the new view’s history
VC1. Replica initiates the view change by sending an accusation against the primary to all replicas.

\[ \langle I - HATE - THE - PRIMARY, v \rangle_{\sigma_1} \]
View Change Subprotocol

VC2.
- Replica receives $f + 1$ accusations that the primary is faulty
- Then it commits to the view change.
VC3.
- Replica receives $2f + 1$ view change messages
- New primary sends new view message
View Change Subprotocol

VC4.
- Replica receives a valid new view message
- Then it sends a view confirmation message to all other replicas.
View Change Subprotocol

VC5:
- Replica receives $2f + 1$ matching `VIEW-CONFIRM` messages
- Begins accepting requests in the new view

Ready for requests!
Implementation Optimizations

• Replacing signatures with MACs
• Separating agreement from execution
• Request batching
• Caching out of order requests
• Read-only optimization
• Single execution response
• Preferred Quorums

Midterm question!
Evaluation
Evaluation: Throughput

\[ f = 1 \text{ tolerated faults} \]

Figure 3

*\[ f = 1 \text{ tolerated faults} \]
Evaluation: Latency

![Latency Graph](image)

Figure 4

Latency per request (ms)

0.0 / 0.0 (r/o)

Q/U

Z

Z

Z

Q/U

Z & PBFT
Evaluation: Fault Scalability

Figure 6

Robust to increasing $f$
Conclusion

“By systematically exploiting speculation, Zyzzyva exhibits significant performance improvements over existing BFT services. ... approaching the theoretical lower bounds for any BFT protocol.”
References


Questions?