Zyzzyva:
Speculative Byzantine Fault Tolerance

R. Kotla, L. Alvisi, M. Dahlin, A. Clement, E. Wong

Sajjad Rahnama, November 1st
Agenda

- Introduction
- Zyzzyva System Model
- Protocol Overview
- Node State and Checkpoints
- Agreement Protocol
- View Change
- Correctness
  - Safety
  - Liveness
Introduction

Byzantine Fault

State Machine Replication

Byzantine Fault Tolerant State Machine Replication
Introduction

PBFT

Practical Byzantine Fault Tolerant Protocol

- $3F+1$ node
- Can Tolerate $f$ faulty node
- 3 Phase
- Pre-Prepare, Prepare, Commit
- 4 One-way messages
Introduction

PBFT

Practical Byzantine Fault Tolerant Protocol

Make sure that I didn’t receive two same sequence number

I know That nobody receive two same sequence number

Everyone know that nobody receive two same sequence number
Introduction

**Zyzzyva**

“A protocol that uses **Speculation** to reduce the cost and **Simplify** the design of BFT state machine replication”
Zyzzyva

- Speculative Execution
- Replies to the client contain *Sufficient history*

**History and response are Stable?**

- **Yes** → Client uses the reply
- **No** → Wait until converge
Introduction

Zyzzyva

• Challenge is ensuring that response to the client become stable
• Move output Commit to the client
• Clients act on request in one or two phases
## Why Zyzzyva?

<table>
<thead>
<tr>
<th>Cost</th>
<th>PBFT</th>
<th>Zyzzyva</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Replicas</td>
<td>3f+1</td>
<td>3f+1</td>
</tr>
<tr>
<td>Replica with application state</td>
<td>2f+1</td>
<td>2f+1</td>
</tr>
<tr>
<td>Critical path 1-way Latency</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>
System Model

Assumptions

- Faulty nodes may behave Arbitrarily
- Faulty nodes cannot break cryptographic signs
- Messages may fail to deliver or delay
Protocol Overview

Subprotocols

Agreement

View Change

Checkpoint
Protocol Overview

Principles and Challenges

• Safety property as they are observed by client
• Replicas can be temporarily inconsistent
• Client detect them, drive them to convergence
• Client rely on consistent responses
• Replicas execute the orders before its Order

Fully Stablished
Protocol Overview

Safety

If a request with sequence number $n$ and history $h_n$ completes, then any request that completes with a higher sequence number $n' \geq n$ has a history $h_{n'}$ that includes $h_n$ as a prefix.

Liveness

Any request issued by a correct client eventually completes.
Protocol Overview

Protocol Communication

Client Send Request to the Primary
Protocol Overview

Protocol Communication

- Primary Forwards the Request to all replicas
- Replicas *Executes* the request
Protocol Overview

Protocol Communication

• Replicas Send Response with history to the client

• \(3f+1\) mutually consistent response then it is done
Protocol Overview

Protocol Communication

- Some of nodes are faulty
- Client Receive between $2f+1$ and $3f+1$ response
Protocol Overview

Protocol Communication

Faulty nodes

- Client Gather $2f+1$ response and make **Commit Certificate**
- Send’s commit certificate to **all nodes**
Protocol Overview

Protocol Communication

Faulty nodes

- Client Respond to CC and **acknowledge** to the Client
- Once **2f+1 acknowledgments** received client act on request
Node State and Checkpoint

- **Ordered History**: History of executed requests
- **Max Commit Certificate**: CC seen by node with the largest seq number
- **Committed History**: History up to seq number of max commit certificate
- **Speculative History**: History follows the committed history
Checkpoint

- A replica constructs a checkpoint every CP_INTERVAL requests.
- Similar to other BFT protocols like PBFT

Reach checkpoint interval → Sign and send CP message to all replicas → 1) Highest #seq of requests 2) digest of current CP → Collect f+1 CP message and done
Node State and Checkpoint

Replica State
Agreement Protocol

Step 1

• Client Sends Request to the Primary

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

• o: operation
• t: timestamp
• c: client Id
Agreement Protocol

Step 2

- Primary receive request and assign seq number
- Forward ordered request to all primary

\langle\langle\text{ORDER-REQ}, v, n, h_n, d, ND\rangle_{\sigma_p}, m\rangle

- \(v\): view number
- \(n\): sequence number
- \(m\): client message
- \(d\): \(H(m)\)
- \(h_n\): \(H(h_{n-1}, d)\)
- \(ND\): application values
Agreement Protocol

Step 3

- Replica receive ordered Request
- Check that:
  - $m$ is wellformed and $d$ is correct digest
  - $n = \max_n + 1$
  - $h_n = H(h_{n-1}, d)$
- Execute the request and create **Spec-Response**
Agreement Protocol

Step 3

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

- \(r\): reply to the operation
- \(i\): replica id
- \(OR\): order request

**Question**: What will happen to out of order Sequence numbers?
Agreement Protocol

Step 3

Out of order Sequence numbers:

- $n \leq \text{max}_n + 1$  
  **Discard the request**

- $n > \text{max}_n + 1$  
  **The replica has some gap in its history**

- Replica send **Fill-Hole** message to the primary
  
  $\langle \text{FILL-HOLE}, v, \max_n + 1, n, i \rangle_{\sigma_i}$

- Primary respond with order request for $k \leq n' \leq n$

**Question**: What will happen if primary doesn’t answer?
Agreement Protocol

Step 3

If primary doesn’t answer to Fill-Hole Message:

• After replica timer for fill-hole message expires replica broadcast Fill-Hole message to all replicas
• Start view change timer
• Replicas which receive Fill-Hole message, will forward Order-Req of corresponding holes to sender if they already have
• If timer expires and still replica doesn’t receive Order-Reqs it will initiate view change
Agreement Protocol

Step 4

Client Gathers Speculative Responses

- Spec-Response messages must match following properties:
  - v: view number
  - n: sequence number
  - c: client id
  - H(r): reply digest
  - h_n: H(h_{n-1}, d)
  - t: request timestamps

Based on number of speculative response and OR four case could happen
Agreement Protocol

Step 4a

• Client Receive $3f+1$ matching response
• It assumes that request is completed
• No acknowledgement will send to replicas
• Replicas cannot determine that request is committed
Agreement Protocol

Step 4b

When some of nodes are faulty:

- Client Receive between $2f+1$ and $3f+1$ matching response
- It assembles $2f+1$ response as a Commit-Certificate
- Send commit message with CC to all replicas

\[
\langle \text{COMM}IT, \ c, \ CC \rangle_{\sigma_c}
\]

CC is the list of all $2f+1$ matching speculative responses
Agreement Protocol

Step 4b-1

- Replica receive a *commit message* from a client containing CC
- Replica acknowledge to the client with *Local-Commit* message
- Send CC to all replicas

1) It already has executed request
   - Send Commit Local

2) It hasn’t execute request
   - Update max sequence number and execute operations and send Commit local message

3) Replica has holes in its history
   - Fill the hole as previously discussed
Agreement Protocol

Step 4b-2

- Client Receive a Local Commit from a 2f+1 replica
- Assume that request is completed
- Send CC to all replicas

**Question**: What will happen if doesn’t receive 2f+1 local-commit?

- It *starts timer* when send commit message
- If timer expires before 2f+1 one local-commit message then it will act same as 4c step
Agreement Protocol

Step 4c

• Client Receive fewer than 2f+1 matching Spec-Response
• It Resend the its request to all Replicas
• Replicas will forward client request to the primary
• A non-primary replica which receive client request
  • 1) If it has cached response it will send that to client
  • 2) if the sequence number is new then send Confirm Message to the primary
Agreement Protocol

Step 4c

• Replica send `Confirm-Message` to primary and ask for `Order-Request`

\[\langle \text{CONFIRM-REQ}, v, m, i\rangle \sigma_i\]

• `m` is client request

• Replica **start timer** after sending `Confirm-Message`

• If primary accepts then it send response to client

• If **timer expires** then it will initiate **view change**
Agreement Protocol

Step 4d

- Client receive response indicating inconsistent ordering by primary
- It sends Proof of Misbehaver to all replicas
- They will initiate view change

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

- Inconsistent Ordering: two spec response with valid OR and view number and different sequence number

\[ \langle \text{POM}, v, POM \rangle_{\sigma_c} \]

Proof of Misbehavior message
View Change

**View Change Sub Protocol**

- Elect new primary
- Must guarantee no change will happen in committed history
- The View Change sub protocol is like previous BFT’s ones
View Change

View Change step 1

- Replica Initiate view change by sending accusation to all replicas

\[ \langle I\text{-HATE\text{-THE\text{-PRIMARY}}, v}\rangle_{\sigma_i} \]

- In previous protocols, this message would indicate that replica is no longer participating in the current view

- This message is only a hint that a replica would like to change views
View Change step 2

- Replica receives $f+1$ accusations that the primary is faulty
- Replica commits to the view change
- No longer participate in current view

$\langle \text{VIEW-CHANGE}, v+1, CC, O, i \rangle_{\sigma_i}$

- Sends view Change message to all replicas
- CC: last commit certificate
- O: ordered request since commit certificate
View Change step 3

- Replica Receives $2f+1$ view change message
- New primary will send \textit{New-View} message to all replicas

$$\langle \text{NEW-VIEW}, v + 1, P \rangle_{\sigma_p}$$

- $P$: is collection of $2f+1$ view change message
- A replica \textit{after sending} view-change message \textit{starts a timer}
- If replicas timer expires it initiate new view change for $v+2$
View Change

View Change step 4

• Replica receives valid New-View Message
• It sends a View-Confirmation Message to all replicas
• The most recent request with a corresponding CC will be accepted as the last committed history
• The most recent request that is ordered subsequent to the CC by at least $f+1$ view-change messages will be accepted.

\[
\langle \text{VIEW-CONFIRM}, v + 1, n, h, i \rangle_{\sigma_i}
\]
View Change

View Change step 5

- New Primary receive $2f+1$ View-Confirm message
- The replica will begin new view
Correctness

Safety

• Show no 2 request with same sequence number
• Show if $n' > n$ is committed then $h$ is prefix of $h'$
• Within a View
  • 3f+1 speculative response or 2f+1 local-commit
  • 1) Correct node send one speculative response
  • 2) Correct node just send local commit after seeing 2f+1 speculative response

• Across Views:
  • In case 2f+1 CC message at least one correct node will send CC in their view change message
  • In case of 3f+1 spec-response every correct replica will include spec response in their view change message
Liveness

- If the primary is correct
  - In case of 3f+1 spec response it will immediately completes
  - In case of 2f+1 spec response because at most f nodes are faulty then it definitely receive 2f+1 local commit
- If the request does not complete during the current view then view change will happen
- If the request does not complete by protocol step 4c client resends request to all replicas
- Any replica that does not receive order-req from primary will send I-Hate-Primary
- There will be f+1 I hate primary or 2f+1 spec response and view change occur or request will complete