Zyzzyva: Speculative Byzantine Fault Tolerance

Junghwan Kim, Dingyu Wang
Motivation
Motivation

- (1) Reduce the cost and
- (2) Simplify the design of
- Byzantine Fault Tolerant (BFT)
  State Machine Replication (SMR)
Review: Byzantine Failure

Byzantine

Omission failures

Crash

Commission failures
Why another BFT protocol?

- Current state-of-the-art is too complex

![Flowchart]

1. **High request contention?**
   - Yes → **PBFT**
   - No → **Low Latency?**

2. **Low Latency?**
   - Yes → **Replication cost < 5f+1?**
     - Yes → **HQ**
     - No → **Q/U**
   - No → **HQ**
Zyzzyva

- Approach the lower bounds in almost every metric
- Simplify the design space of BFT replicated service
Main Idea
Review: State Machine Replication

1. Make server deterministic (state machine)
2. Replicate server
3. Ensure that all replicas go through the same sequence of state transitions
4. Vote on replica outputs
Traditional SMR

Agree

Execute
Zyzzyva: Speculative SMR

Execute

Verify
Traditional SMR

- Request
- Agreement
- Execution
- Reply

- Replicas execute requests in agreed order
- Client receives replies in the consistent order
Zyzzyva: Speculative SMR

- Replicas execute requests without agreement
- Client commits output only if system is consistent
Zyzzyva Details and Protocol
Zyzzyva Details

- Speculative Execution and Leverage Client!
- Client verifies reply’s **history information**
  
  a. Client **commits the output** if stable with unanimous replies
  b. Client **transmits commit certificate** if stable replies with some failures
  c. Client **invokes view change** if unstable replies
Zyzzyva: History information

The state maintained by each replica

- Garbage collected history
- Committed history
- Speculative history

Checkpoint interval

- Committed checkpoint
- Tentative checkpoint
- Committed snapshot
- Tentative snapshot

Max Commit Certificate

Requires $f + 1$ matching checkpoint messages to be committed
Zyzzyva: Agreement Protocol

Request: $R_c$

Step 1. Client sends request to the primary
Zyzzyva: Agreement Protocol

- Step 2. Primary receives request, assigns sequence number and forwards ordered request to replicas.
Zyzzyva: Agreement Protocol

Step 3. Replica receives ordered request, speculatively executes it, and responds to the client
Zyzzyva: Agreement Protocol

Request: \( R_c \)

\( \langle R_c, k \rangle \)

\( \langle R_{ik}, H_{ik} \rangle \)

Speculative Execution

\( \triangleright \) Step 4a. Client receives 3f+1 matching responses and completes the request

\( 3f+1 \)

\( R_{1k} = R_{2k} = \ldots, H_{1k} = H_{2k} = \ldots \)
Zyzzyva: Agreement Protocol

2f+1~3f
\(R_{1k} = R_{2k} = \ldots, H_{1k} = H_{2k} = \ldots\)

Request: \(R_c\)

\(<R_c, k>\)

\(<R_{ik}, H_{ik}>\)

Speculative Execution

◇ Step 4b. Client receives between 2f+1 and 3f matching responses
Zyzyzyva: Agreement Protocol

2f + 1 ~ 3f
R_{1k} = R_{2k} = \ldots, H_{1k} = H_{2k} = \ldots

Request: R_c

C: <H_{1k}, H_{2k}, H_{3k}>

<_{R_c, k}>

<_{R_{ik}, H_{ik}}>

Speculative Execution

Commit

- Step 4b.0. Client assembles and transmits a commit certificate to the replicas
Zyzzyva: Agreement Protocol

Step 4b.1. Replica receives a commit message from a client containing a commit certificate and acknowledges with a local-commit message.
Zyzzyva: Agreement Protocol

2f+1 \sim 3f
R_{1k} = R_{2k} = \ldots, H_{1k} = H_{2k} = \ldots

C: <H_{1k}, H_{2k}, H_{3k}>

Request: \( R_c \)

\(<R_c, k>\)

\(<R_{ik}, H_{ik}>\)

Speculative Execution

Commit

\( \text{Step 4b.2. Client receives a local-commit messages from 2f+1 replicas and completes the request} \)
Zyzzyva: Agreement Protocol

\[
< 2f+1
R_{1k}=R_{2k}=\ldots, H_{1k}=H_{2k}=\ldots
\]

Request: \( R_c \)

\[< R_{c, k}> \]

\[< R_{ik}, H_{ik}> \]

Speculative Execution

◊ Step 4c. Client receives fewer than 2f+1 matching responses
Zyzzyva: Agreement Protocol

< 2f+1
R_{1k} = R_{2k} = ..., H_{1k} = H_{2k} = ...

Request: $R_c$

Speculative Execution

◊ Step 4c.0. Client resends its request to all replicas
Zyzzyva: Agreement Protocol

\[ <2f+1, R_{1k} = R_{2k} = \ldots, H_{1k} = H_{2k} = \ldots > \]

Request: \( R_c \)

Speculative Execution

✧ Step 4c.1. Replicas forward the request to the primary to ensure the request is assigned a sequence number and eventually executed
Zyzzyva: Agreement Protocol

< 2f+1
R_{1k} = R_{2k} = \ldots, H_{1k} = H_{2k} = \ldots

Request: R_c

< R_{c, k}>

Speculative Execution

◊ Step 4c.2. If request is returned, go to step 3
◊ Otherwise, initiate a view change
Zyzzyva: Agreement Protocol

- Request: $R_c$
- $<R_c, k>$
- $<R_{ik}, H_{ik}>$
- Speculative Execution

- Step 4d. Client receives responses indicating inconsistent ordering by the primary
- (May co-occur with 4b or 4c)
Zyzzyva: Agreement Protocol

- **Request**: $R_c$
- **Proof of misbehavior**: $<R_{ik}, R_{ik}>$
- $R_{ik} \neq R_{jk}$

**Speculative Execution**

- **Step 4d.0.** Client sends a proof of misbehavior to the replicas, which initiate a view change to oust the faulty primary.
Zyzzyva: View Change

- VC1. Replica initiates the view change by sending an accusation against the primary to all replicas. (i–hate–primary message)
- VC2. Replica receives $f + 1$ accusations that the primary is faulty and commits to the view change. (view–change message)
- VC3. Replica receives $2f + 1$ view change messages.
Zyzzyva: View Change

纪律 4. 复制接收到一个有效的新的视图消息，并发送一个视图确认消息给所有其他副本。

纪律 5. 复制接收到 $2f + 1$ 个匹配的视图确认消息，并开始接受在新视图中的请求。
View-Change: The Case of Missing Phase

Traditional view change

- $f$ replicas cannot finish change view
- $f + 1$ replicas cannot complete requests

Zyzzyva view change

- View change do not stall unless view change is certain
- Zyzzyva shifts the costs needed to deal with a faulty primary from the critical path to the view change sub-protocol

1. Make request
2. Agree
3. Get response

Exchange what they know in third phase
View Change: The Case of Uncommitted Request

- Case 4b. $2f + 1 \sim 3f$ matching responses
  - NEW-VIEW message from new primary includes all commit certificates

- Case 4a. $3f + 1$ matching responses
  - VIEW-CHANGE messages from correct replicas include all request messages $<R_c, k>$
  - New primary incorporates all requests that appear in at least $f + 1$ of the $2f + 1$ VIEW-CHANGE messages
Zyzzyva Correctness
Correctness: Safety within a view

- For any request completed,
- Case 4a. 3f+1 matching responses
- Case 4b. 2f+1 local-commit messages
- Due to the quorum property, no two requests complete with the same sequence number
- For any two completed requests, one history is a prefix of the other history
Correctness: Safety across views

- For any request completed in the previous view,

- Case 4a. $3f+1$ matching responses
  - at least $f+1$ among $2f+1$ view–change message contains this information

- Case 4b. $2f+1$ local–commit messages
  - at least $f+1$ correct replicas received a commit certificate
  - at least 1 among $2f+1$ view–change message contains this information
Correctness: Liveness

- If the primary is correct and a correct client issues the request, then the request completes.
- A client will receive at least 2f + 1 of response messages → completes 4a or 4b.
- If a request from a correct client does not complete during the current view, then a view change occurs.
- Case 4c. At least 1 correct replica initiates view change.
- Case 4d. Inconsistency → view change.
Optimizations
Optimizations

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

(*) also in PBFT (#) from previous papers
Zyzzyva5
Making the faulty case fast

1
f
f
f
f

Request: $R_c$

$\langle R_c, k \rangle$

$\langle R_{ik}, H_{ik} \rangle$

- use 2f additional witness replicas for a total of 5f+1 replicas
- Complete in a single phase with f faulty replicas
Evaluation
Figure 3: Realized throughput for the 0/0 benchmark as the number of client varies for systems configured to tolerate $f = 1$ faults.
Fig. 4: Latency for 0/0, 0/4, and 4/0 benchmarks for systems configured to tolerate $f = 1$ faults.
Fault Scalability

Figure 6: Fault scalability: Peak throughputs
Fault Scalability

Batch size = 1

Bottleneck server cryptographic operations

Batch size = 10
Fault Scalability

Batch size = 1

Batch size = 10

Bottleneck server messages
Discussion
Discussion

- Zyzzyva exploits the speculation systematically.
- Zyzzyva achieves high performance in terms of both throughput and latency (approaching theoretical minimum).
- Great explanation of main ideas, details and performance at the same time.