Zyzzyva: Speculative BFT

Bryan Liu
Introduction

- Replicas *speculatively* execute requests *without* running an agreement protocol to reduce computation cost and simplify design of BFT SMR
  - Replicas’ states can temporarily diverge
- The *client* is responsible for detecting and fixing replica divergence
  - The client only acts on the reply if that state is stable i.e. will eventually commit to all replicas
Motivation

- **Query/Update (Q/U)** - Low latency if contention is low and $5f + 1$ replicas is acceptable
- **Hybrid Quorum (HQ)** - Scales well assuming no batching
- **Practical Byzantine Fault Tolerance (PBFT)** - Handles high contention and supports message batching
Protocol

- Protocol split into three sub-protocols
  - Agreement
  - View Change
  - Checkpoint

**SAF** 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.

**LIV** Any request issued by a correct client eventually completes.
Protocol

- Each replica maintains . . .
  - An ordered *history* of its executed requests
  - A copy of the *max commit certificate* (CC)
    - Sequence numbers covered by this max CC are *committed history*
    - Sequence numbers *not* covered are *speculative history*
  - A *stable checkpoint* and *stable application state snapshot* every *CP_INTERVAL* requests
    - Generated from a *tentative checkpoint* and *application state snapshot*
    - History begins from the committed checkpoint
  - A *response cache* containing the latest request and response of each client
Agreement

1. Client sends request to the primary

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

2. Primary receives request, assigns sequence number, and forwards ordered request to replicas

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

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

<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c)</td>
<td>Client ID</td>
</tr>
<tr>
<td>(CC)</td>
<td>Commit certificate</td>
</tr>
<tr>
<td>(d)</td>
<td>Digest of client request message</td>
</tr>
<tr>
<td>(d = H(m))</td>
<td></td>
</tr>
<tr>
<td>(i, j)</td>
<td>Server IDs</td>
</tr>
<tr>
<td>(h_n)</td>
<td>History through sequence number (n)</td>
</tr>
<tr>
<td>(h_n = H(h_{n-1}, d))</td>
<td></td>
</tr>
<tr>
<td>(m)</td>
<td>Message containing client request</td>
</tr>
<tr>
<td>(max_n)</td>
<td>Max sequence number accepted by replica</td>
</tr>
<tr>
<td>(n)</td>
<td>Sequence number</td>
</tr>
<tr>
<td>(o)</td>
<td>Operation requested by client</td>
</tr>
<tr>
<td>(OR)</td>
<td>Order Request message</td>
</tr>
<tr>
<td>(POM)</td>
<td>Proof Of Misbehavior</td>
</tr>
<tr>
<td>(r)</td>
<td>Application reply to a client operation</td>
</tr>
<tr>
<td>(t)</td>
<td>Timestamp assigned to an operation by a client</td>
</tr>
<tr>
<td>(v)</td>
<td>View number</td>
</tr>
</tbody>
</table>
Agreement

3. Replica receives ordered request, speculatively executes it, and responds to the client

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

**Output:** $\langle \langle \text{SPEC} - \text{RESPONSE}, v, n, h_n, H(r), c, t \rangle_{\sigma_i}, i, r, OR \rangle$

- Accept the message if . . .
  - $m$ is well-formed and $d$ is correct
  - $n = \max_n + 1$
  - $h_n = H(h_{n-1}, d)$
- Append request to history, execute the request using its own state to produce $r$
Agreement

3. Replica receives ordered request, speculatively executes it, and responds to the client

- If a hole is introduced ($n > max_n + 1$), discard the message and send $\langle FILL - HOLE, v, max_n + 1, n, i \rangle_{\sigma_i}$ to the primary and sets a timer to receive ORDER-REQ messages between sequence number $max_n + 1$ and $n$
- If the timer expires, broadcast FILL-HOLE to other replicas and initiate a view change

<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$</td>
<td>Client ID</td>
</tr>
<tr>
<td>$CC$</td>
<td>Commit certificate</td>
</tr>
<tr>
<td>$d$</td>
<td>Digest of client request message $d = H(m)$</td>
</tr>
<tr>
<td>$i,j$</td>
<td>Server IDs</td>
</tr>
<tr>
<td>$h_n$</td>
<td>History through sequence number $n$ $h_n = H(h_{n-1}, d)$</td>
</tr>
<tr>
<td>$m$</td>
<td>Message containing client request</td>
</tr>
<tr>
<td>$max_n$</td>
<td>Max sequence number accepted by replica</td>
</tr>
<tr>
<td>$n$</td>
<td>Sequence number</td>
</tr>
<tr>
<td>$o$</td>
<td>Operation requested by client</td>
</tr>
<tr>
<td>$OR$</td>
<td>Order Request message</td>
</tr>
<tr>
<td>$POM$</td>
<td>Proof Of Misbehavior</td>
</tr>
<tr>
<td>$r$</td>
<td>Application reply to a client operation</td>
</tr>
<tr>
<td>$t$</td>
<td>Timestamp assigned to an operation by a client</td>
</tr>
<tr>
<td>$v$</td>
<td>View number</td>
</tr>
</tbody>
</table>
4. Client gathers speculative responses

**Inputs:** \( \langle \langle SPEC - RESPONSE, v, n, h_n, H(r), c, t \rangle_{\sigma_i}, i, r, OR \rangle \)

- Input messages *match* if all fields except *OR* and *i* fields are identical
- 4 cases to consider...
Agreement

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

- The request and its history are **complete**. Deliver $r$ to the application
  - The protocol guarantees all correct replicas will execute this request eventually across view changes
Agreement

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

Output: $\langle \text{COMMIT}, c, CC \rangle_{\sigma_c}$

- Commit certificate consists of
  - List of $2f + 1$ replicas, their signatures, and their signed portions in their SPEC-RESPONSE
- For efficiency, only need one SPEC-RESPONSE message

<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$</td>
<td>Client ID</td>
</tr>
<tr>
<td>$CC$</td>
<td>Commit certificate</td>
</tr>
<tr>
<td>$d$</td>
<td>Digest of client request message $d = H(m)$</td>
</tr>
<tr>
<td>$i, j$</td>
<td>Server IDs</td>
</tr>
<tr>
<td>$h_n$</td>
<td>History through sequence number $n$ $h_n = H(h_{n-1}, d)$</td>
</tr>
<tr>
<td>$m$</td>
<td>Message containing client request</td>
</tr>
<tr>
<td>$max_n$</td>
<td>Max sequence number accepted by replica</td>
</tr>
<tr>
<td>$n$</td>
<td>Sequence number</td>
</tr>
<tr>
<td>$o$</td>
<td>Operation requested by client</td>
</tr>
<tr>
<td>$OR$</td>
<td>Order Request message</td>
</tr>
<tr>
<td>$POM$</td>
<td>Proof Of Misbehavior</td>
</tr>
<tr>
<td>$r$</td>
<td>Application reply to a client operation</td>
</tr>
<tr>
<td>$t$</td>
<td>Timestamp assigned to an operation by a client</td>
</tr>
<tr>
<td>$v$</td>
<td>View number</td>
</tr>
</tbody>
</table>
Agreement

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

Output: \(\langle \text{LOCAL} - \text{COMMIT}, v, d, h, i, c \rangle_{\sigma_i}\)

- Replica should execute the request with the specified sequence number and history in the current view.
  - If replica’s history matches, update the max commit certificate if the received CC’s sequence number is greater and send \(\langle \text{LOCAL} - \text{COMMIT}, v, d, h, i, c \rangle_{\sigma_i}\) to client \(c\).
  - If replica’s history does not match:
    - If local history differs exclusively due to holes, send FILL-HOLE message to primary.
    - Initiate a view change.
Agreement

4b.2. Client receives LOCAL-COMMIT messages from $2f + 1$ replicas and completes the request.

- If the client does not receive $2f + 1$ LOCAL-COMMITs before a timeout, the client performs step 4c
Agreement

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

- For non-primary replicas . . .
  - If the request’s timestamp is at most client c’s entry in the response cache, send cached response back to client.
  - If the request’s timestamp is higher, send \(\text{CONFIRM} - \text{REQ}, v, m, i\) to primary and start a timer. If primary responds and the replica receives the ORDER-REQ, process it like in step 3. Otherwise initiate a view change.
Agreement

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

- For the primary . . .
  - Upon receiving CONFIRM-REQs, if the request is new send an ORDER-REQ with a new sequence number. Else send the cached ORDER-REQ from the response cache.
Agreement

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

- Only considers ORDER-REQ entry in SPEC-RESPONSE
- Misbehavior = Same request and same view with differing sequence number or history
- Send \( (POM, v, POM)_{σc} \) to all replicas, which upon receiving will initiate a view change
View Change

● The VIEW-CHANGE message includes
  ○ A new view
  ○ Sequence number of the replica’s most recent stable checkpoint
  ○ A replica’s set of commit certificates since the latest stable checkpoint

● Zyzzyva differs in . . .
  ○ Adding a new phase to the view change sub-protocol
  ○ Weakening the condition under which a request is included in the history of the NEW-VIEW message
View Change – the Missing Phase

- Once a replica decides view change, it will stop accepting messages from the previous view.

- Consider the scenario:
  - $f$ faulty replicas, one of which is the primary.
  - The faulty primary never communicates with $f$ correct replicas.
  - The faulty replicas never respond to the client or issue a view change.

- **Solution:** do not abandon a view unless it is guaranteed that all correct replicas will also trigger a view change.
  - Add a new phase and new message called I-HATE-THE-PRIMARY before VIEW-CHANGE.
View Change – Uncommitted Requests

- Only clients may know when and if a request has completed
  - On a view change, how do we know if a replica’s history is safe?

- Case 1: Commit phase is run
  - At least $f + 1$ correct replicas have a commit certificate for response $r$
  - We can handle this like in PBFT with prepare certificates

- Case 2: Only speculation phase is run
  - No commit certificate will be generated
  - **Solution:** 1) Add all ORDER-REQ messages since the last stable checkpoint or commit certificate. 2) Primary also extends its history if it receives consistent ORDER-REQs that appear in at least $f + 1$ view change messages
Optimizations

- Replace key signatures with MACs
- Separate agreement from execution by partitioning the replicas
- Batch concurrent requests
- Cache out-of-order requests
- Send read requests directly to replicas
- Respond to client with a single full response
- Increase the number of replicas to minimize latency when replicas are faulty
Experiments

- Compared no replication, PBFT, HQ, Q/U, Zyzzyva, Zyzzyva5
- Used 3.0 GHz Pentium-4 machines with Linux 2.6 kernel
- Tolerate $f = 1$ faults unless otherwise noted
- Benchmarks: 0/0, 0/4, 4/0
  - $j/k$ – Client sends a request of size $j$ KB and receives a reply of size $k$ KB
Experiments - Throughput

- Zyzzyva outperforms PBFT since it has lower crypto and message overhead
- Q/U and HQ do not support batching
Experiments - Latency

- Q/U (ideal) : Best-case implementation assuming low contention
- r/o : read-only requests. Tests Zyzzyva with read optimization
Experiments - Load
Experiments – Fault Scalability

[Graphs and charts showing experimental results for fault scalability, including throughput and cryptographic operations under different batch sizes.]