Zyzzyva: Speculative Byzantine Fault Tolerance

Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin,
Allen Clement, and Edmund Wong
Dept. of Computer Sciences University of Texas at Austin

Presented by:
Seongwoo Choi, David Haddad,
Blake McMurray, Sairamvinay Vijayaraghavan

UCDAVIS
COMPUTER SCIENCE

Agenda

Introduction

Abstract

Explanation (Why? System Model)

Protocol (Principles and Challenges)

Agreement Protocol

View Change Protocol

Evaluation

Optimizations

Discussion

Conclusion

Abstract & Introduction

Uses ***speculation*** to reduce the cost of ***BFT replication***

– Primary replica proposes order of client requests to all secondary replicas (standard)

– Secondary replicas ***speculatively*** execute the request without going through an agreement protocol to 3 validate that order (New)

Zyzzyva

Previously:

– States of correct replicas may diverge

– Replicas may send diverging replies to client

Zyzzyva’s solution

– Clients detect inconsistencies

– Help convergence of correct replicas to a single total ordering of requests

– Replicas have **stable checkpoints** and corresponding stable app. state snapshot.

– Reject inconsistent replies

Byzantine Generals' Problem

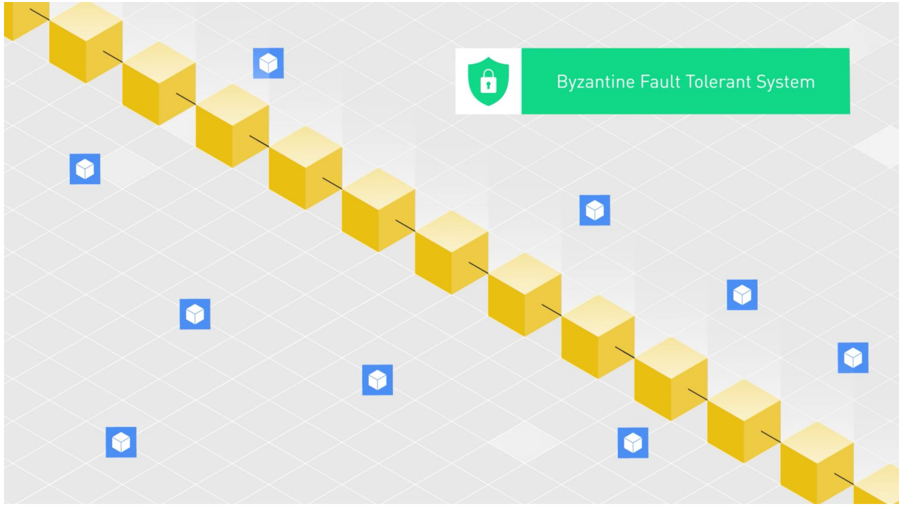
https://youtu.be/VWG9xcwjxUg?t=53
Automatically starts at 0:52
Stop at 2:21

Byzantine Fault Tolerance

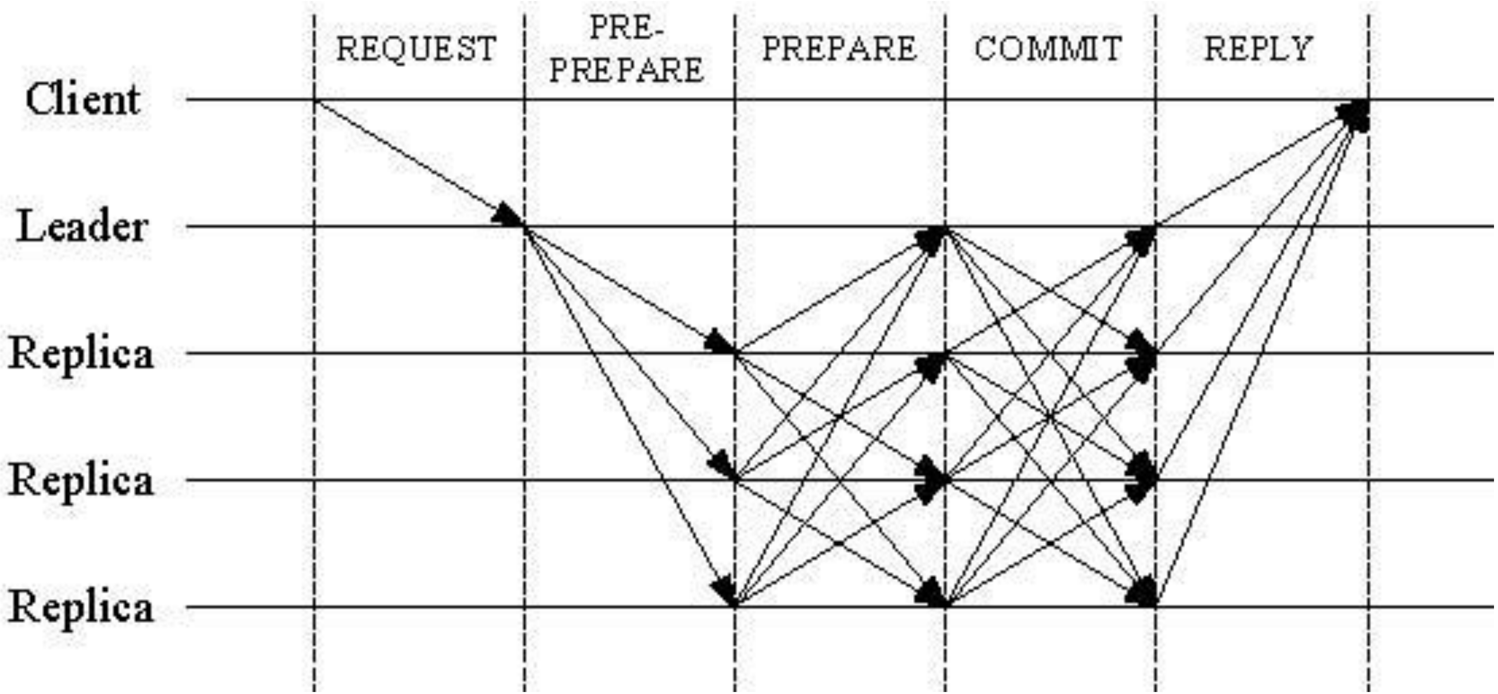
A property of system that is able to resist class of failures derived from Byzantine Generals' Problem.

It is able to continue operating even if some of the nodes fail to communicate or act maliciously.

There are many possible ways to build BFT systems: related to different types of consensus algorithms.



Practical Byzantine Fault Tolerance



Three Trends make BFT replication increasingly attractive

– Increasing value of data and decreasing cost of hardware

– BFT is becoming cheaper

– Cost of 3 -way non-BFT replication close to cost of BFT replication

Why Another BFT Protocol?

“The state of the art for BFT state machine replication is distressingly complex”

Zyzzyva simplifies the design space of BFT replicated services by approaching the lower bounds in almost every key metric.

Throughput: both use batching when load is high and thereby approach the lower bound on the number of authentication operations

Latency: executes requests in three one-way message delays

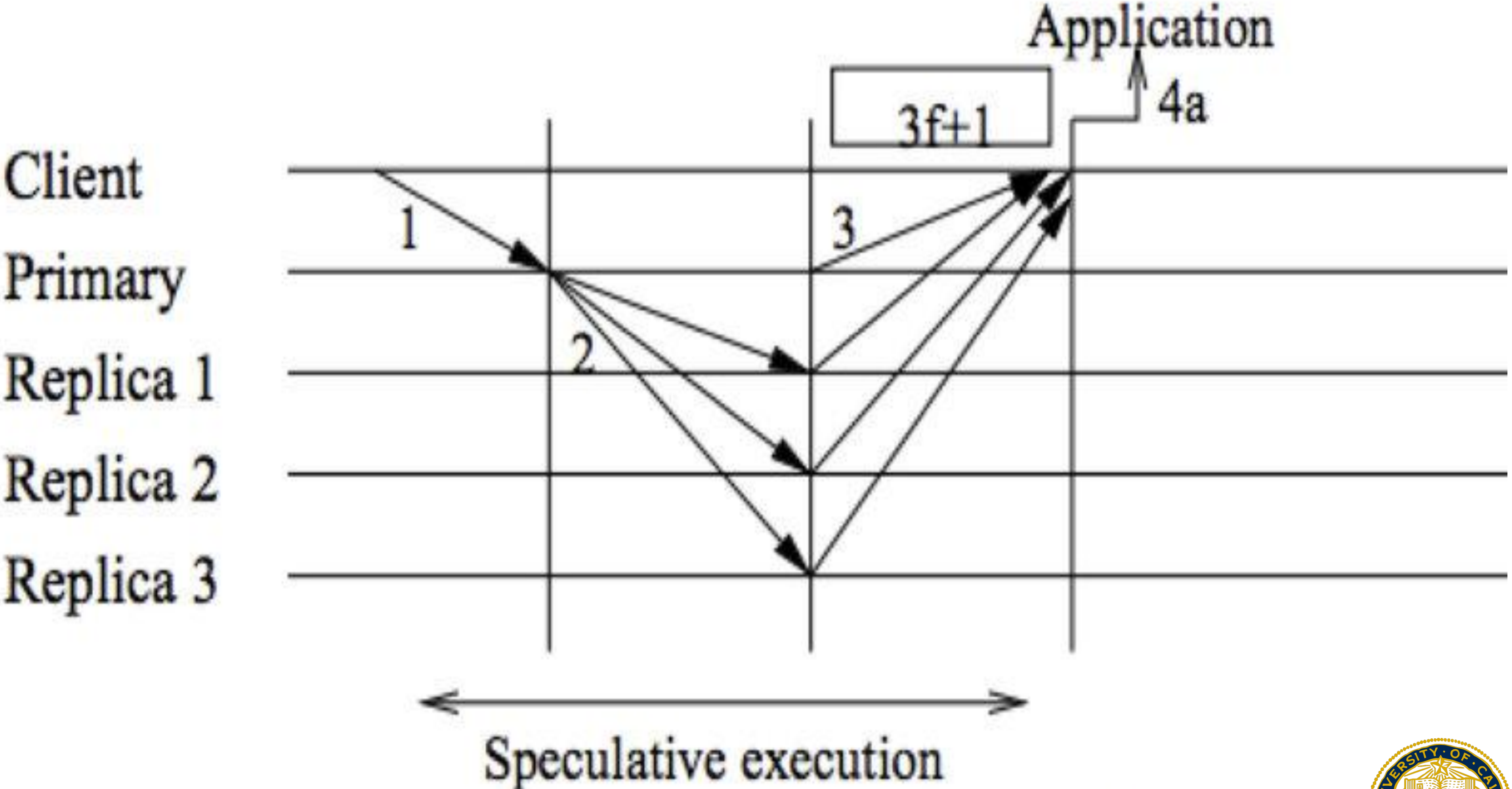
Scalability: the metrics that depend on f grow as slowly or more slowly in Zyzzyva

System Model

Safety properties hold in any asynchronous distributed system.

Implements a BFT service using ***state machine replication***.

Services limit the damage done by Byzantine clients by authenticating clients, enforcing access control to deny clients access to objects.



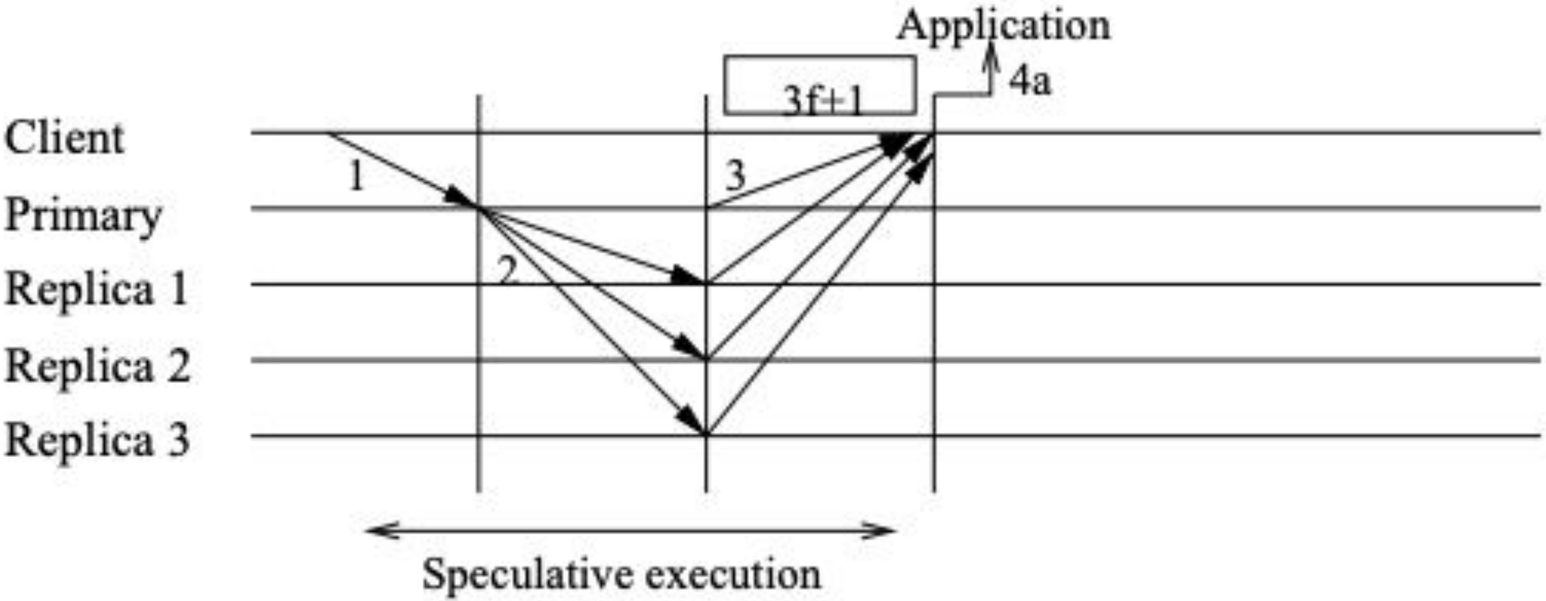
(a) Gracious execution

Protocol

The protocol consists of 3 subprotocols:

1. **Agreement**: Orders requests for execution by the replicas
2. **View Change**: Coordinates the election of a new primary when current is faulty or non responsive
3. **Checkpoint**: Limits state that must be stored by replicas and reduces the cost of performing view changes

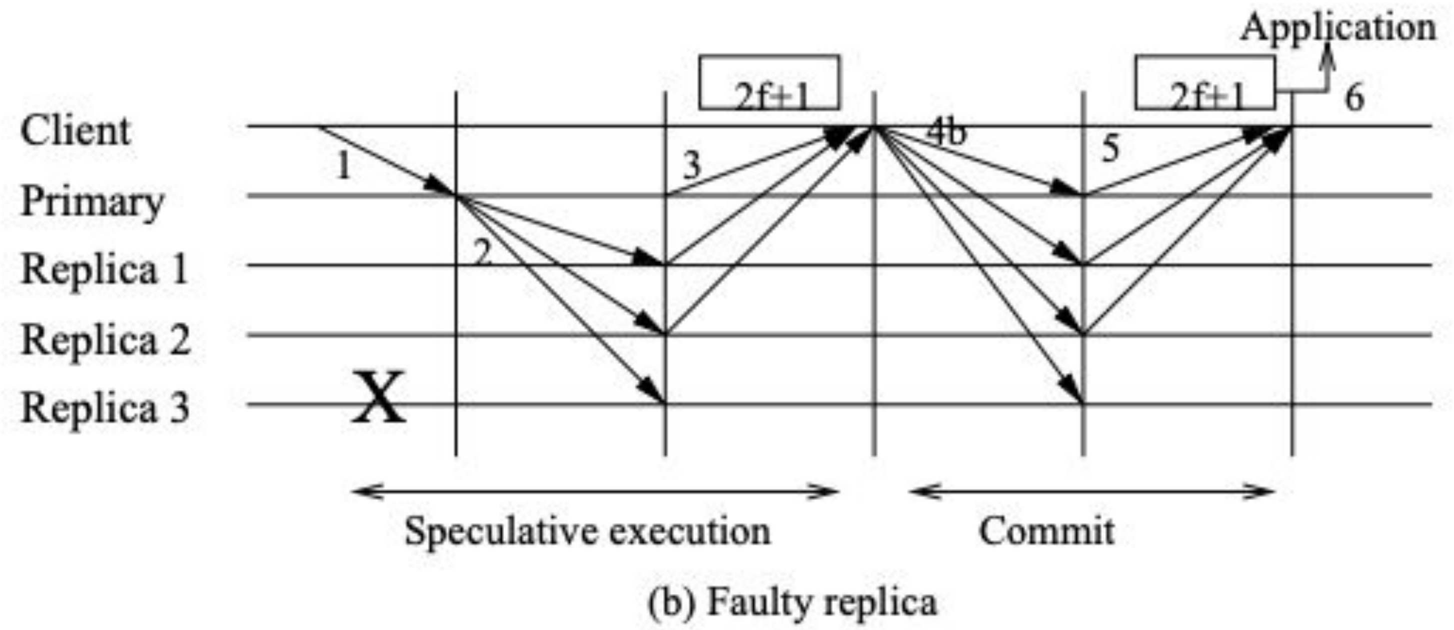
Zyzzyva Speculative Execution: 2 Cases



(a) Gracious execution

Case 1: Client receives **mutually consistent** responses from all nodes in the network (or at least 3f+1 for some predefined f value)

Zyzzyva Speculative Execution: 2 Cases



Case 2: Client receives responses from < 3f + 1 but > than 2f + 1 replicas

Attempt At Explaining 2f+1 Replicas

When faults in the network are due to **fail-stop** nodes only (no maliciousness)

1. Assume a network of n nodes, f of which are faulty
2. Then the system must function correctly with n - f nodes
3. If we assume the network only needs a simple majority (> 50% of votes) to reach consensus, then the functioning nodes (n - f) must outnumber the faulty nodes (f)
4. Put in algebraic terms:

$$(n - f) > f \rightarrow n > 2f \rightarrow n \geq 2f + 1$$

Attempt At Explaining 3f+1 Replicas

When faults in the network are **byzantine** (**fail-stop**, **software errors**, or **maliciousness)**

1. Again, let's assume that a network containing n nodes has f byzantine nodes
2. Out of the n nodes, f of them can either be unresponsive (fail-stop) or malicious/software errors
3. If f nodes are unresponsive (i.e they are fail-stop), then we would only need 2f + 1 nodes
4. But if the f nodes are byzantine, then they can cast malicious votes. The worst case scenario is when the f nodes all perform maliciously. We need the majority of the nodes to be honest under that circumstance
5. Put in algebraic terms:

$$(n - f) - f > f \rightarrow n > 3f \rightarrow n \geq 3f + 1$$

What is The Structure of The Blockchain at Each Replica?

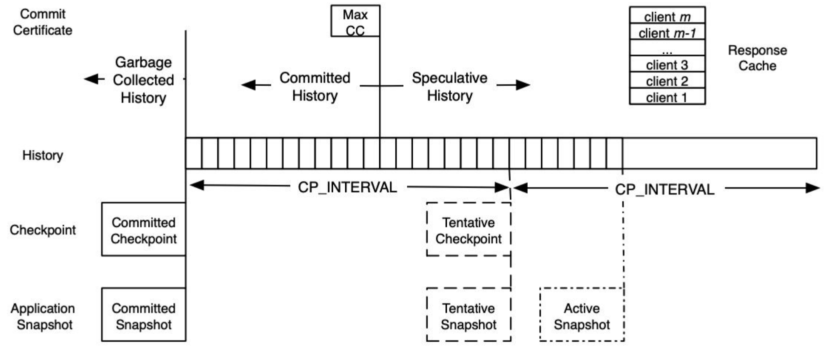
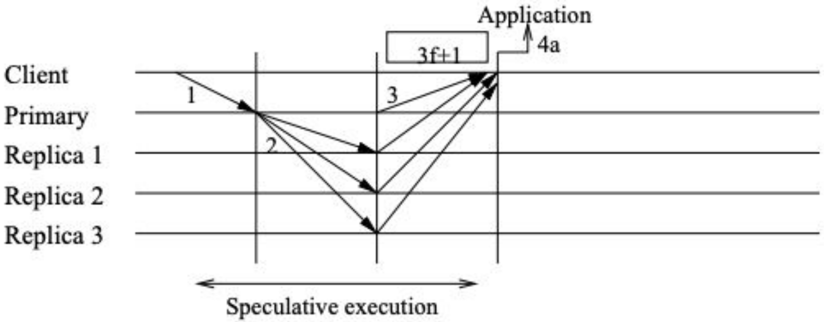


Figure 2: State maintained at each replica.



Each replica maintains 2 'histories': The committed history (which has been verified by f + 1 replicas) and the speculative history, which has not been confirmed

Every CP_INTERVAL # of requests, the replica generates a tentative checkpoint and snapshot and sends a signed message to all replicas. These checkpoints and snapshots become stable when the replica receives f + 1 corresponding signed messages from other replicas

Agreement Protocol

Agreement Protocol

- **Consensus** between the clients and replicas and the primary

- Ensure
  - **Clients** respond on replies that correspond to **stable requests**

  - The **protocol** must ensure **execution** of stable requests eventually **commit** at all correct servers.

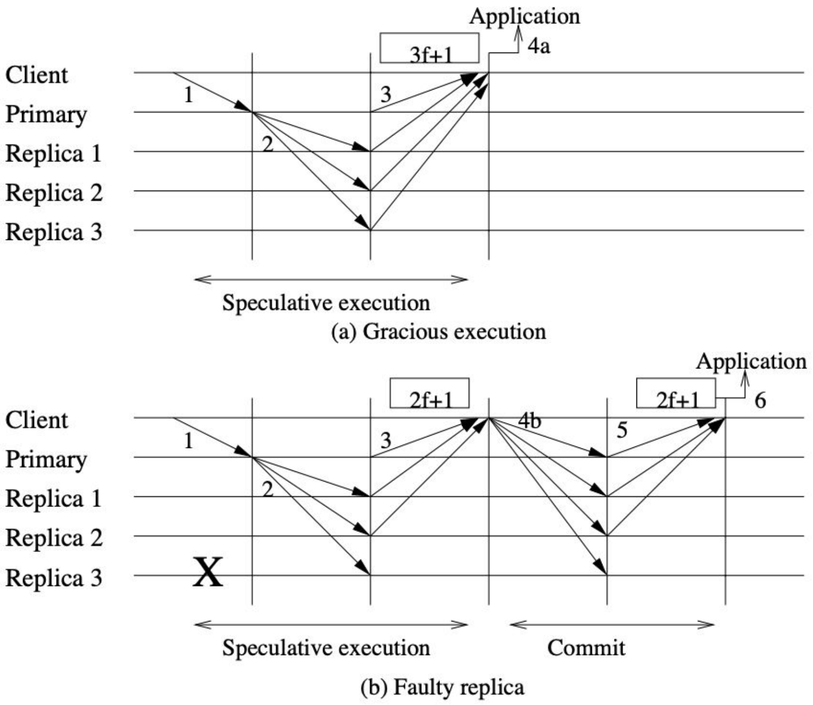
- **Defines** the **rules** a server uses to process each **message**.
- Principle: **committed** at all correct **replicas** (servers) with the same **sequence number** and **history** serving requests made by client.

Agreement Protocol Working

- Request completes at a client
    - when the client receives **3f + 1** matching responses

    - **acknowledgements** from 2f +1 replicas that they have received a commit certificate comprising a local commit from 2f + 1 replicas

- It handles various challenges
    - Missed messages
    - Faulty clients
    - Faulty primary



Message Related Terms

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

Agreement Protocol Working-1

- Client sends request to primary (treating it like another replica)
    - **REQUEST** messages are sent from the client, along with
        - Client ID c
        - Timestamp t
        - Operation o
- Primary accepts request and forwards it to all replicas.
    - Accepts only when the **timestamp** is larger than current largest timestamp noted by client
    - Message m is received **directly** from the **client** (<REQUEST, o, t, c>)
    - **View** v is observed and the **sequence** n is assigned.
    - The message ‘m’ is **digested** (into ‘d’) and history h_n is obtained
    - **Non-deterministic application** (e.g: locks and timeouts) are also relayed (ND)
    - **Final Message**: <<ORDER-REQ,v, n, h_n, d, ND>, m>

Agreement Protocol Working- Replica receives the request

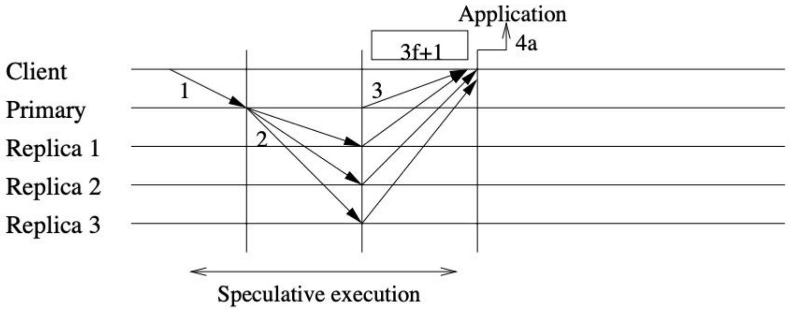
- Accepts **<<ORDER-REQ, v, N, H_n, d, ND>, m>** when
  - ‘m’ is well formed message
  - $N == Max_n + 1$
  - Histories match
- It **includes** and **speculatively executes** this accepted request from primary.
- Replies with the **message: <<SPEC-RESPONSE, v, n, hn, H(r) , c, t>, i, r, OR>** where OR is order request received (excluding m).

- Fill Hole messages
  - Whenever there’s a missing sequence number assigned information lost.
  - Gets the required **fill-hole request** delivered by **primary** within the **timer** set
    - If **timer ends,** there is a **VIEW-CHANGE** protocol started

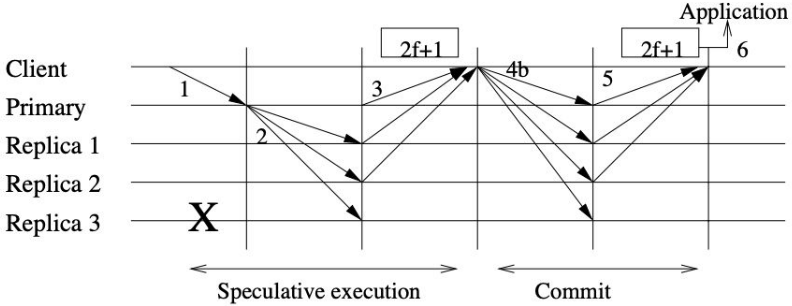
Agreement Protocol Working- Client Receives Speculative Responses

- There can be more than one same message based on
  - Same view (v)
  - Same History (h)
  - Same sequence number (n)
  - Same timestep (t)
  - Same Client (c)

- 4 different cases of handling
  - **3f + 1** speculative responses received
  - **Between 2f+1 and 3f+1** responses received
  - **Lesser than 2f+1** responses received
  - Responses with **inconsistent** orderings



(a) Gracious execution



(b) Faulty replica

Agreement Protocol Working - Client receives at least 2f + 1 responses

- **3f + 1 responses** received
  - Completed request and nothing assumed to be lost from replica
  - Reply delivered directly to the application

- **Between 2f + 1 and 3f + 1** responses received
  - Client sends a **commit message**: <COMMIT, c, CC> to all replica
  - CC has **all** the received **2f + 1 replicas** and the **SPEC-RESPONSE** messages.
  - After sending the commit message
    - Replica receives it, checks **local history consistency** and sends **LOCAL-COMMIT** message.
    - Client receives the **LOCAL-COMMIT from 2f + 1 replicas** and completes the request.

Agreement Protocol Working - Remaining Cases

- Client receives lesser than 2f + 1 responses
  - Happens most probably because of **faulty primary**
  - **Client** sends a broadcast (REQUEST) message directly to all replica (2f+1).
  - **Replica**: depending on timestamp, returns either a cached response or it requests primary.
  - **Primary**: If request is new, send new ORDER-REQ message using next sequence number.

- Inconsistent Ordering of responses
  - Client issues a **Proof of Misbehavior (POM)** suspecting primary is faulty
  - This is sent to **all replicas**
  - VIEW-CHANGE Protocol is initiated once this is received
  - Considers on the **Order-Req field only** since it is based on sequence number and history.

View Change Protocol

View Change Protocol

- Guarantees of electing a **new primary**.

- It requires a **correct replica**
  - To observe the **primary as faulty**
  - Possess **evidence** that **f+1 replicas have committed** to a View-Change.

- Traditional methods (Byzantine) needs to **maintain the history** of the requests.

- Additional properties of View-Change Protocol in Zyzzyva
  - **Liveness**
  - **Safety**

Liveness: Missing Phase

- **Strengthens** commitment of correct replica by adding **“I hate new Primary” phase**.
- Liveness brings “Prepare” and “Commit” phase into one single phase (following PBFT).
- Every **correct** replica has to abandon a view for view-change.

- Procedure
  - A correct replica uses **voting system (no-confidence votes)** across other correct replica to identify faulty primary.
  - If atleast **f+1** votes, then View-Change is started to change the primary.
  - They use the rule **I-Hate-New-Primary** rule to elect for a faulty primary.

Safety: Uncommitted Request

- Weakens when request appears in the history included in the message.
- In case of **2f + 1** responses received by client
    - **Correct replica** has at least f+1 commit certificates
    - VIEW-CHANGE will contain all the commit certificates received from a LOCAL-COMMIT.

- In case of **3f + 1** responses received by client
    - **No commit** certificate in this case
    - All ORDER-REQ messages are saved in all the correct replicas.
    - **New Primary** would include all the ORDER-REQ messages into its history.

- There is no case where 2 completed requests can have the **same sequence number** by correct replica. This is a benign case of preserving safety.

Section 3: Correctness

1. **Safety**:
   a. **Agreement protocol is safe within single view**
      i. No Two requests complete with the same sequence number n
      ii. h_n is a prefix of $h_{n'}$ for n < n' and completed request r and r'
   b. **Agreement and view protocol together ensure safety across views**

2. **Liveness**: **Zyzzyva guarantees liveness only during periods of synchrony**
   a. If **primary** is **correct** when a correct client issues the request, then the **request completes**
   b. If a request from a correct client **does not complete** during the **current view** then the **view change does not complete** during the **current view** then a view change occurs

Evaluation

Evaluation

The protocol was evaluated on the following:

1. **Throughput**
2. **Latency**
3. **Fault Scalability**
4. **Performance During Failures**

Protocols to be compared

Query/Update (Q/U)

Hybrid-Quorum replication (HQ Replication)

Practical Byzantine Fault Tolerance (PBFT)

Zyzzyva

Throughput

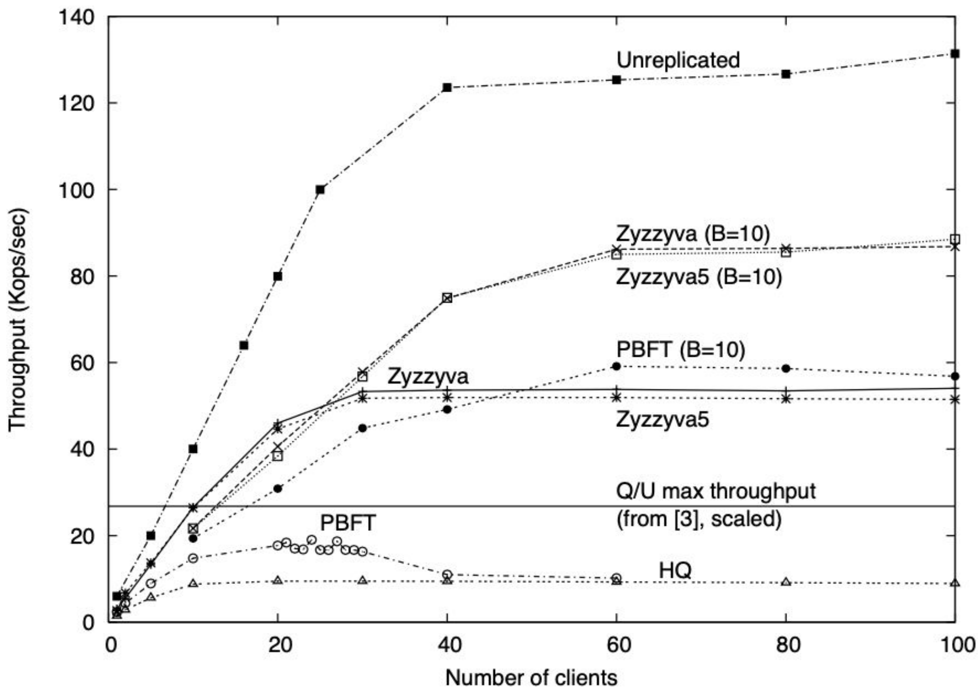


Figure 3: Realized throughput for the 0/0 benchmark as the number of client varies for systems configured to tolerate $f = 1$ faults.

Latency

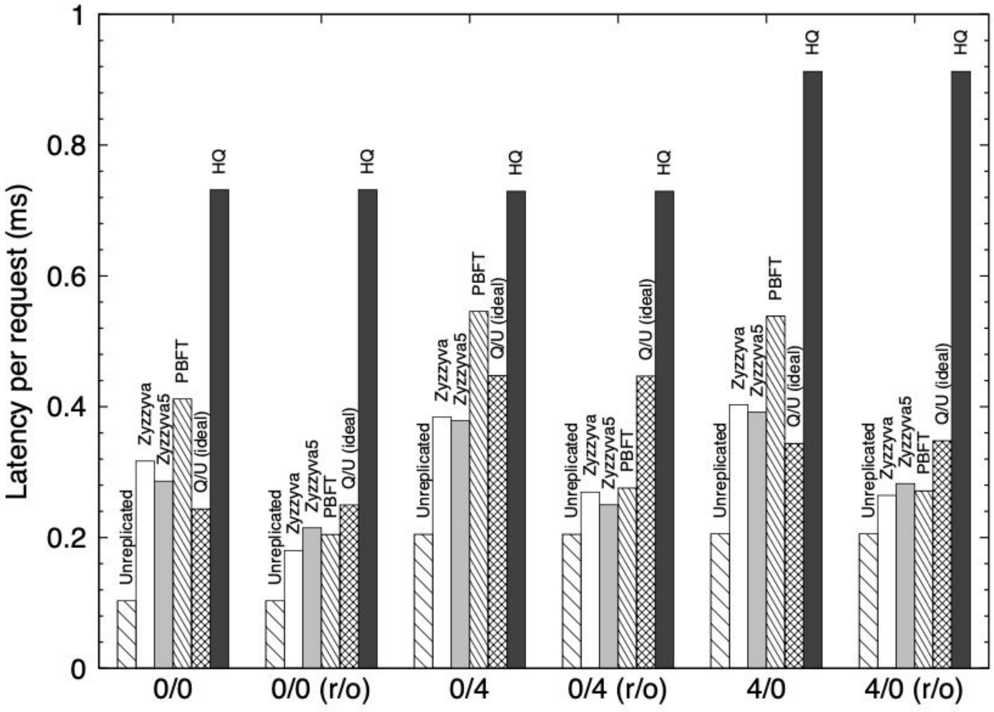


Figure 4: Latency for 0/0, 0/4, and 4/0 benchmarks for systems configured to tolerate $f = 1$ faults.

Latency vs Throughput

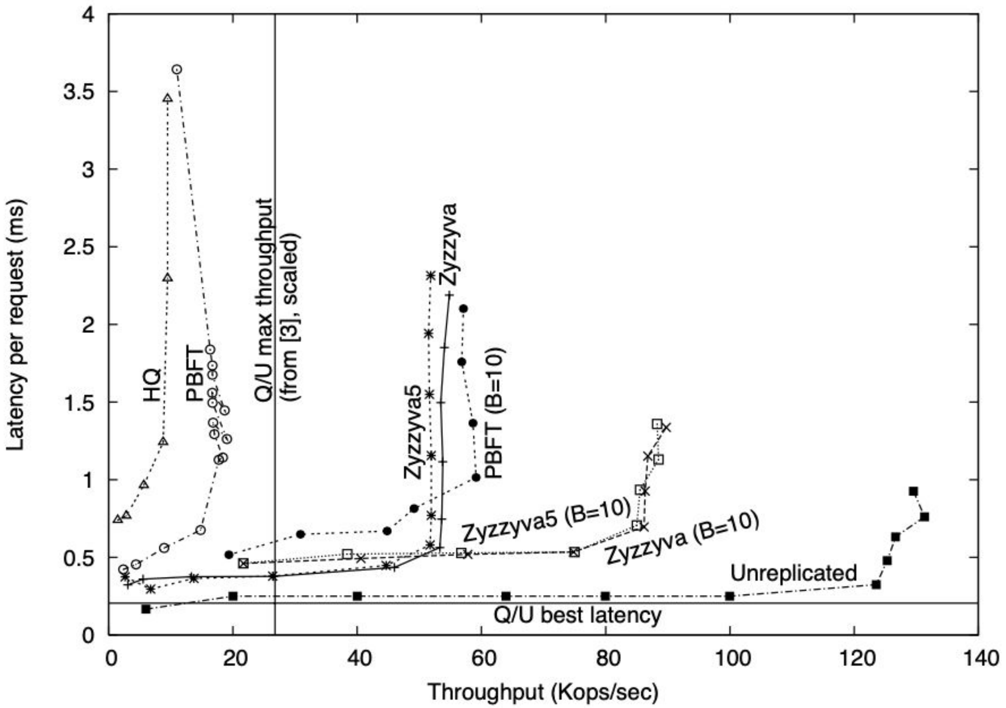


Figure 5: Latency vs. throughput for systems configured to tolerate $f = 1$ faults.

Fault Scalability

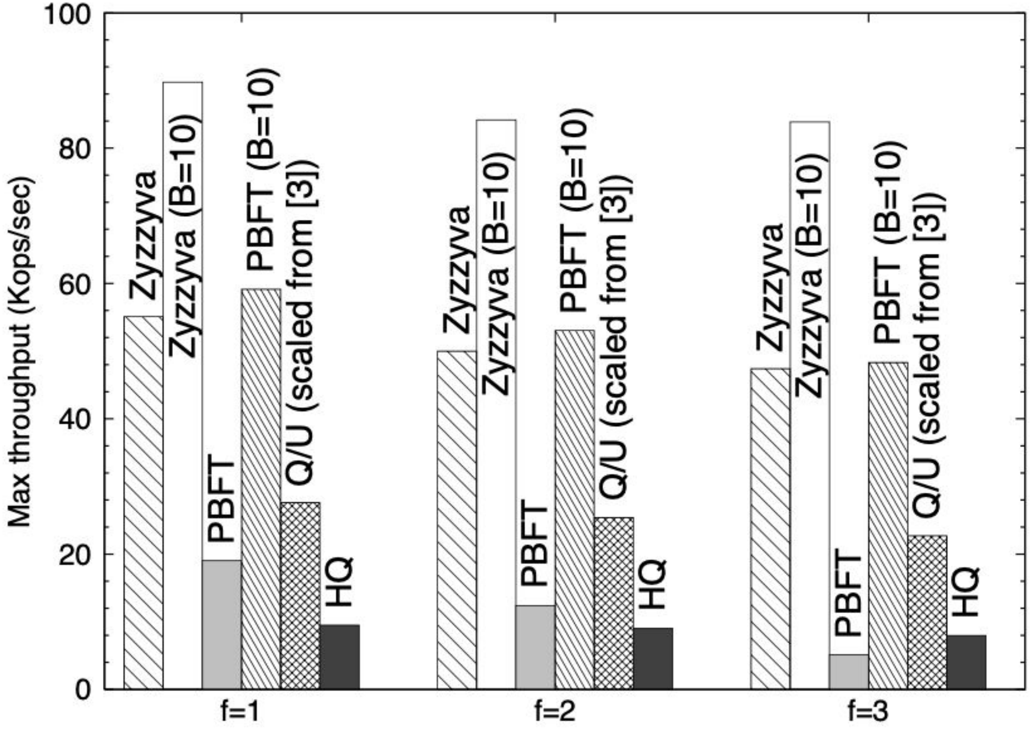
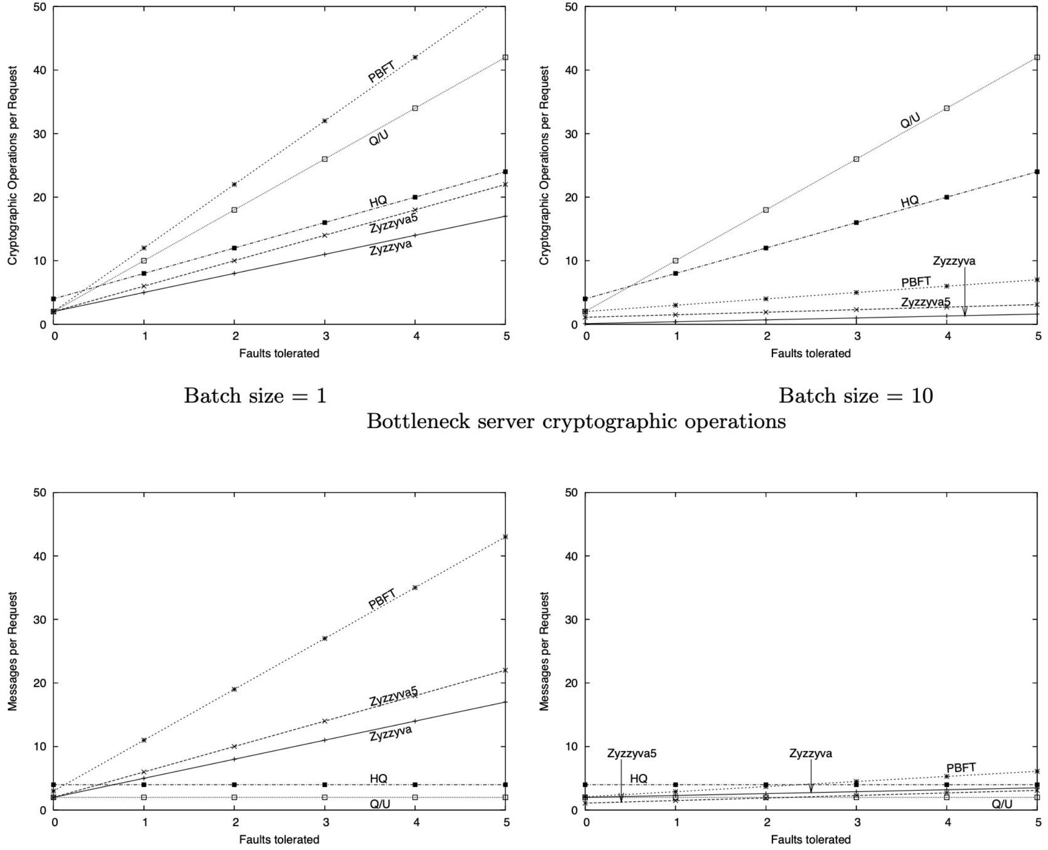


Figure 6: Fault scalability: Peak throughputs

Fault Scalability



Batch size = 1

Batch size = 10

Bottleneck server cryptographic operations

Batch size = 1

Batch size = 10

Bottleneck server messages

Figure 7: Fault scalability using analytical model

Performance During Failure

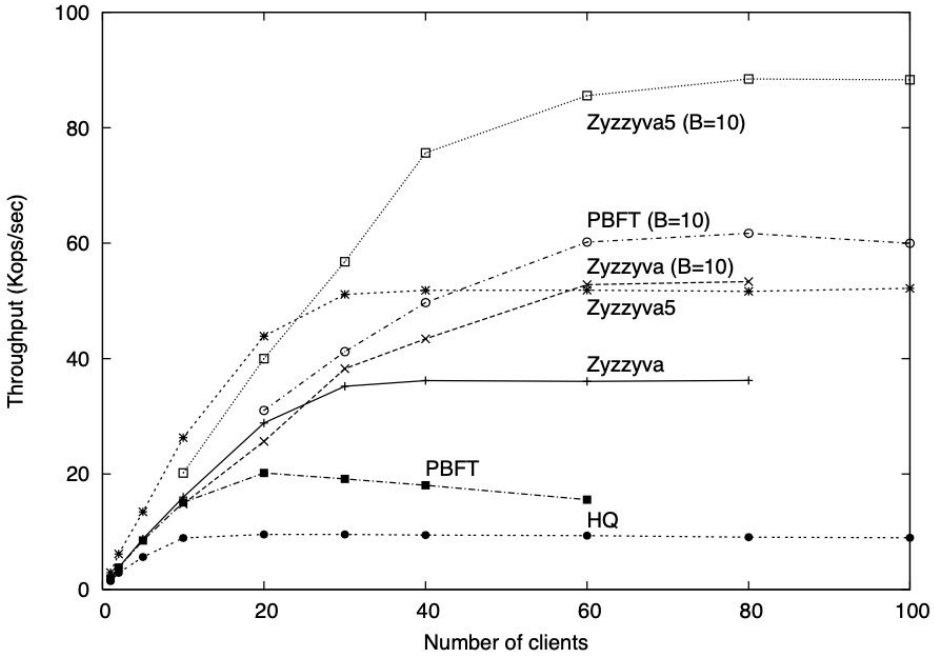


Figure 8: Realized throughput for the 0/0 benchmark as the number of client varies when f non-primary replicas fail to respond to requests.

Implementation Optimizations

Implementation Optimizations

The protocol has 7 optimizations that were applied:

1. **Replacing Signatures with MACS**
2. **Separating Agreement from Execution**
3. **Request Batching**
4. **Caching Out of Order Requests**
5. **Read-Only Optimization**
6. **Single Execution Response**
7. **Preferred Quorums**

Discussion

Zyzzyva Uses Speculation

- Reduce cost of BFT
- Simplify design of BFT

Zyzzyva Advantages over QU (Query Update)

1. **Fewer** replicas
2. Improved throughput via **batching**
3. **Simpler** State Machine Replication Semantics
4. Ability to support **high** contention **workloads**

Zyzzyva Disadvantages over QU

1. Will this actually **resolve** the problem?
2. What if there is an **extra** work?
3. Simpler does **not** mean it works **all** the **time**.

Q/U sidesteps this lower bound by providing a slightly weaker service than state machine replication and optimizes cases without concurrent access to any state

Zyzzyva vs PBFT

Zyzzyva reduces cryptographic overheads and increases peak throughout by a factor of two to an order of magnitude for demanding workloads.

References:

[CS198.2x Week 1] Byzantine Fault Tolerance - YouTube

ZYZZYVA SPECULATIVE BYZANTINE FAULT TOLERANCE R Kotla L (slidetodoc.com)

What is Byzantine Fault Tolerance | Explained For Beginners - YouTube

https://www.youtube.com/watch?v=Uj638eFIWg8&ab_channel=MITVideoProductions

https://youtu.be/dfsRQyYXOsQ

The End

Zyzzyva: Speculative Byzantine Fault Tolerance