State Machine Replication for the Masses with BFT-SMaRt

Hsin-Yang Huang
Chih-shang Chen

Outline

1. Introduction

2. BFT-SMART Design

3. Implementation

4. Alternative Configuration

5. Evaluation

6. Lessons Learned

7. Conclusion

Introduction

Reason:

1.PBFT’s architecture does not fully exploit modern hardware

2.UpRight exhibits a performance significantly lower than other systems.

Characteristic:

1.Java-based

2.high-performance and correctness

3.support reconfigurations of the replica

Design principles

- Tunable fault model
  - non-malicious Byzantine-faults
  - malicious Byzantine-faults
  - Simplified SMR protocol
- Simplicity
  - emphasis on protocol correctness
  - avoid optimizations that could bring extra complexity

Design principles

- Modularity
  - uses a well defined consensus primitive in its core
  - easy to implement and reason about

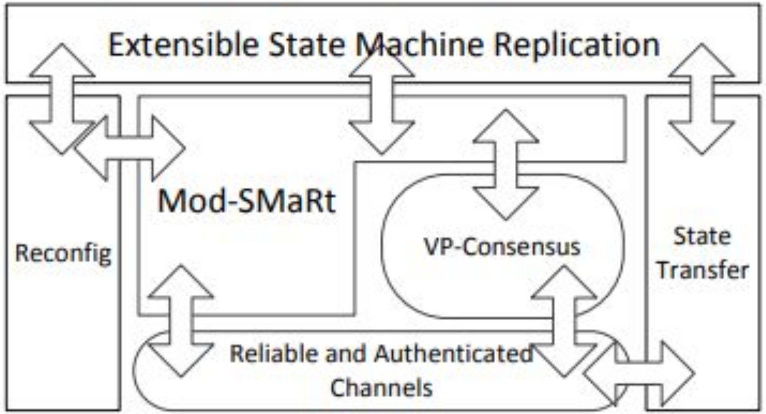

Extensible State Machine Replication
Mod-SMaRt
Reconfig
VP-Consensus
State Transfer
Reliable and Authenticated Channels

Figure 1. The modularity of BFT-SMaRt.

Design principles

- Simple and extensible application programming interface
  - Provide simple API such us *invoke(command)* and *execute(command)*
  - Implemented using a set of alternative calls, callbacks or plug-ins (if API not support some methods)
- Multi-core Awareness
  - Take advantage of multi-core architecture of servers

System model

Configuration:

1. $n \geq 3f+1$ to tolerate Byzantine faults
2. $n \geq 2f+1$ to tolerate Crash faults
3. reconfigure replicas at runtime

Links:

1. message authentication code(MAC) over TCP/IP
2. Symmetric keys for replica-replica channel
3. Optional signed request for client-replica channels.

Core protocol

- Total order multicast
  - During normal execution, clients send their requests to all replicas and wait for their replies
  - Total order is achieved through consensus protocol

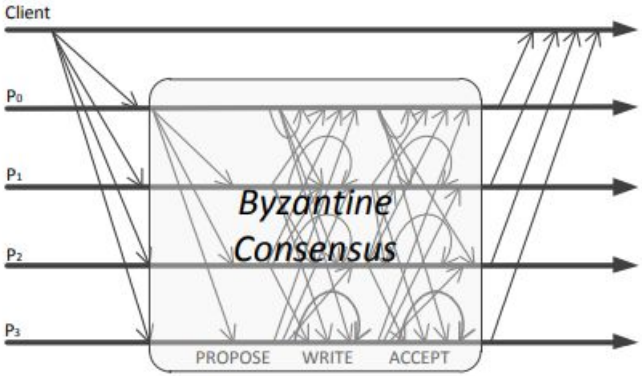


Figure 2. BFT-SMaRt normal phase message pattern.

Core protocol (con’t)

State Transfer

- to log batches of operations in a single disk
- take snapshots at different points of the execution in different replicas
- perform state transfer in a collaborative way

Core protocol (con’t)

Reconfiguration:

- Initiated by *View Manager* client
- Must be signed with a special private key
- *View Manager* sends a special message to the replica that is waiting to be added or removed from the system informing the replica.

Implementation

1.Staged message processing
2.Bounded queue

Netty thread

- Check unordered or ordered request
- Verify client's request

Proposer thread

- Assemble a batch of requests
- Transmitting the PROPOSE message

Sender thread

- Serialize message and produce a MAC
- Send it using TCP sockets

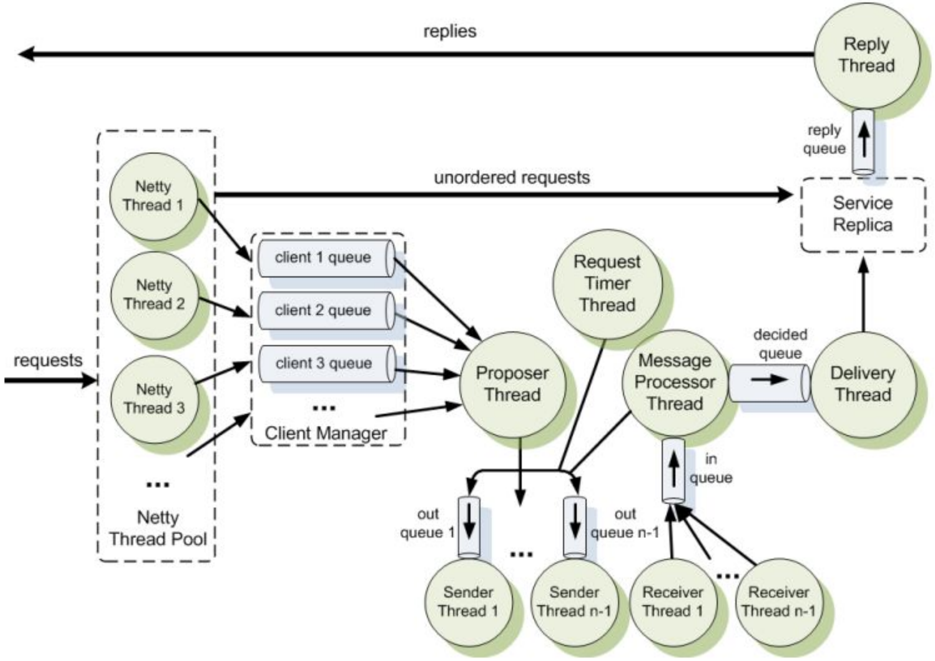


Figure 3. BFT-SMaRt replica staged message processing.

Implementation

Receiver thread

- Deserialize message
- Put it on the inqueue

Message processor thread

- Fetch messages from the inqueue
- Process message if they belong to current consensus stage
- Put finished decided batch on decide queue

Delivery thread

- Remove request on client queue
- Invoke service replica to generate replies

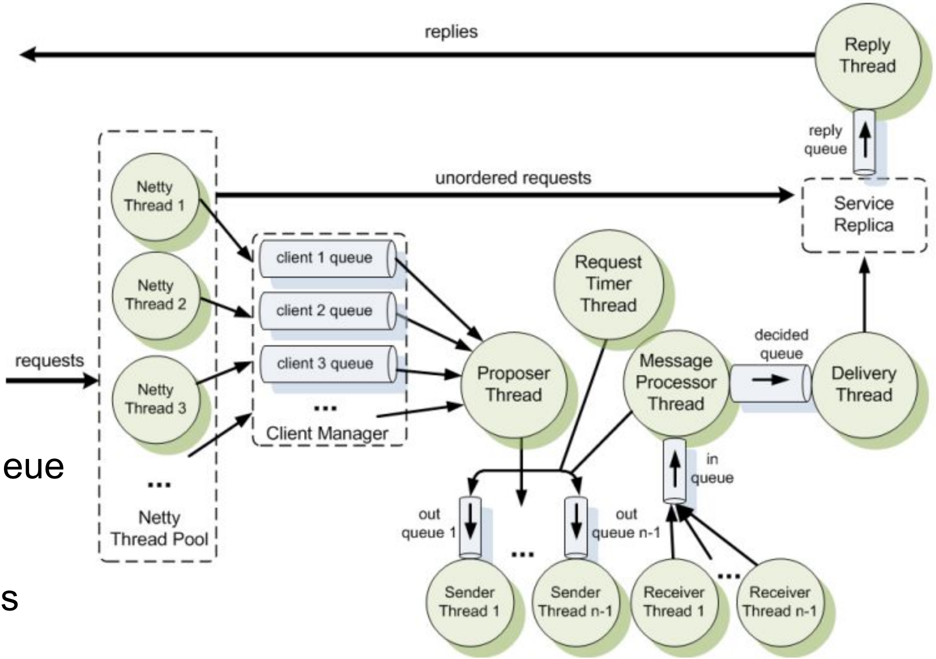


Figure 3. BFT-SMaRt replica staged message processing.

Implementation

Reply thread

- Fetch request from reply queue
- Send it back to client

Request timer thread

- Activated periodically to verify
  If some requests remained more
  Than a predefined time.

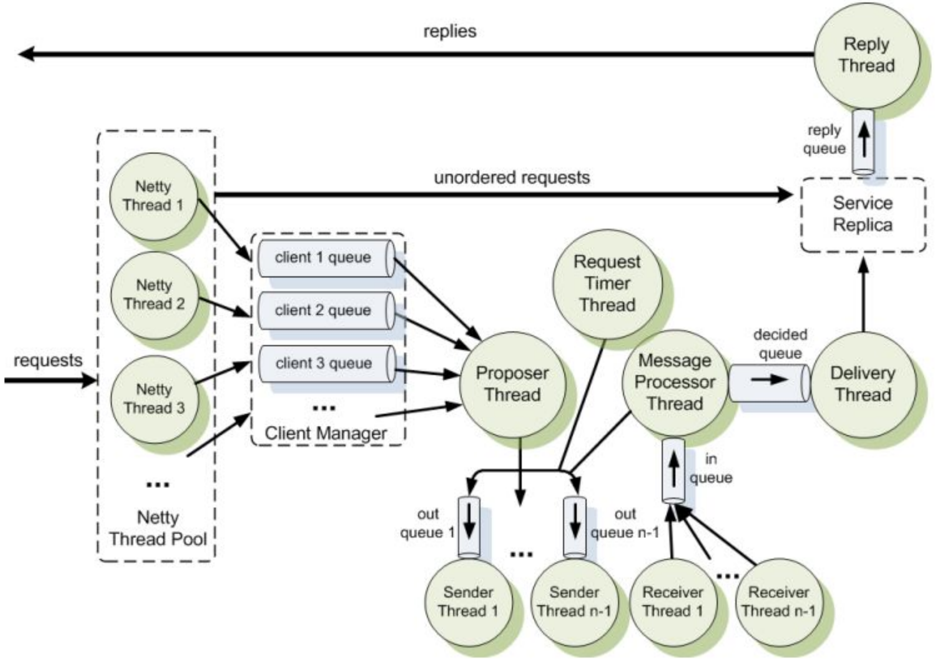


Figure 3. BFT-SMART replica staged message processing.

Alternative Configurations

1. Crash Fault Tolerance (CFT)

   Every node that *do not give a reply* is assumed to be in a crashed state.

   Tolerance: $f < n/2$ (simple minority)

   Sol => bypass WRITE step

2. Malicious Byzantine Faults

   Malicious leader to lasuch undetectable attacks.

   Sol => periodic leader changes

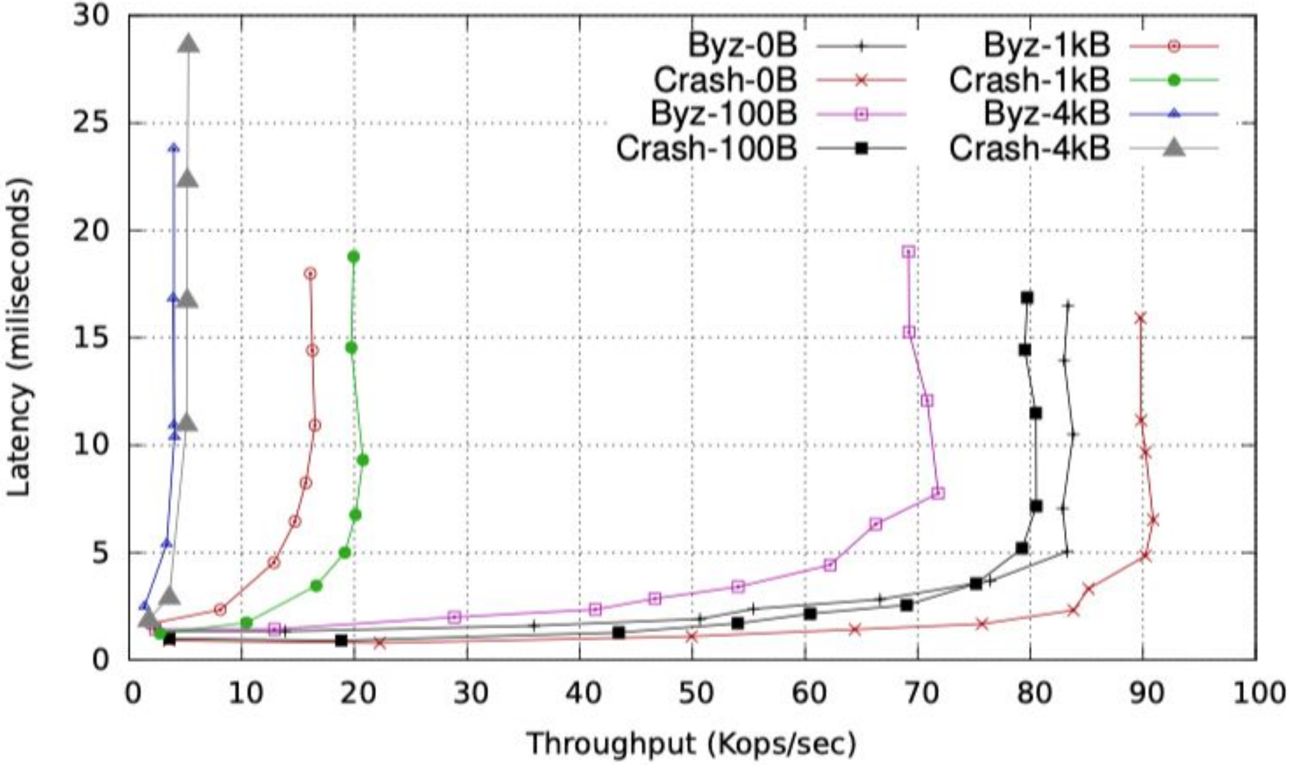
Evaluation

1. Raw throughput and Latency

2. Performance in different systems

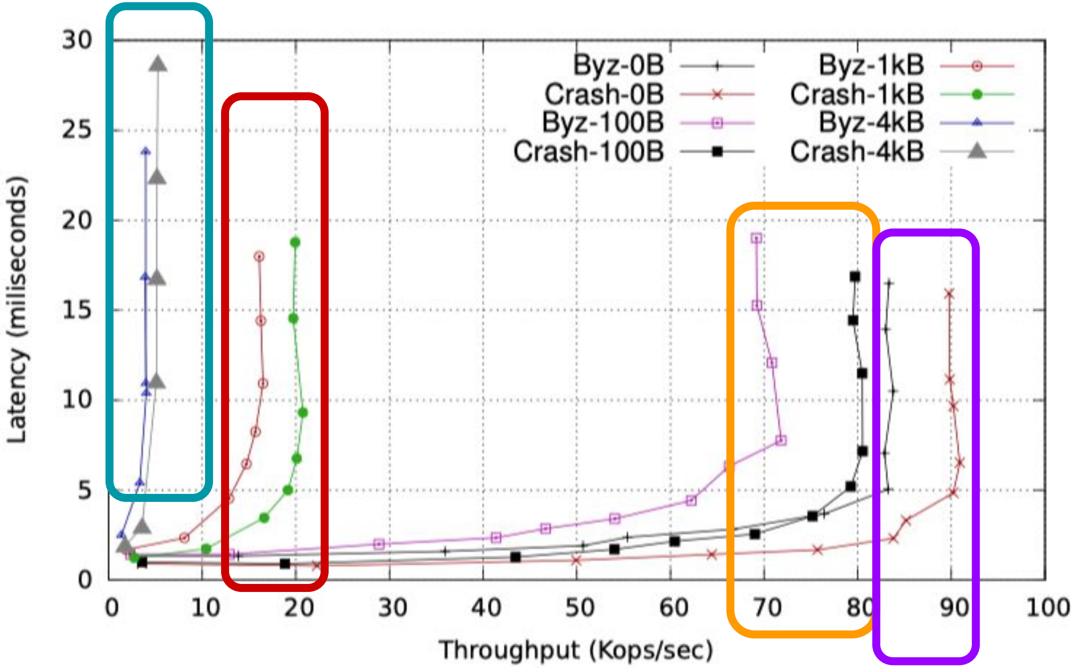
3. The performance of a BFT-SMART-based system when withstanding faults and reconfiguration.

Raw Throughput and Latency

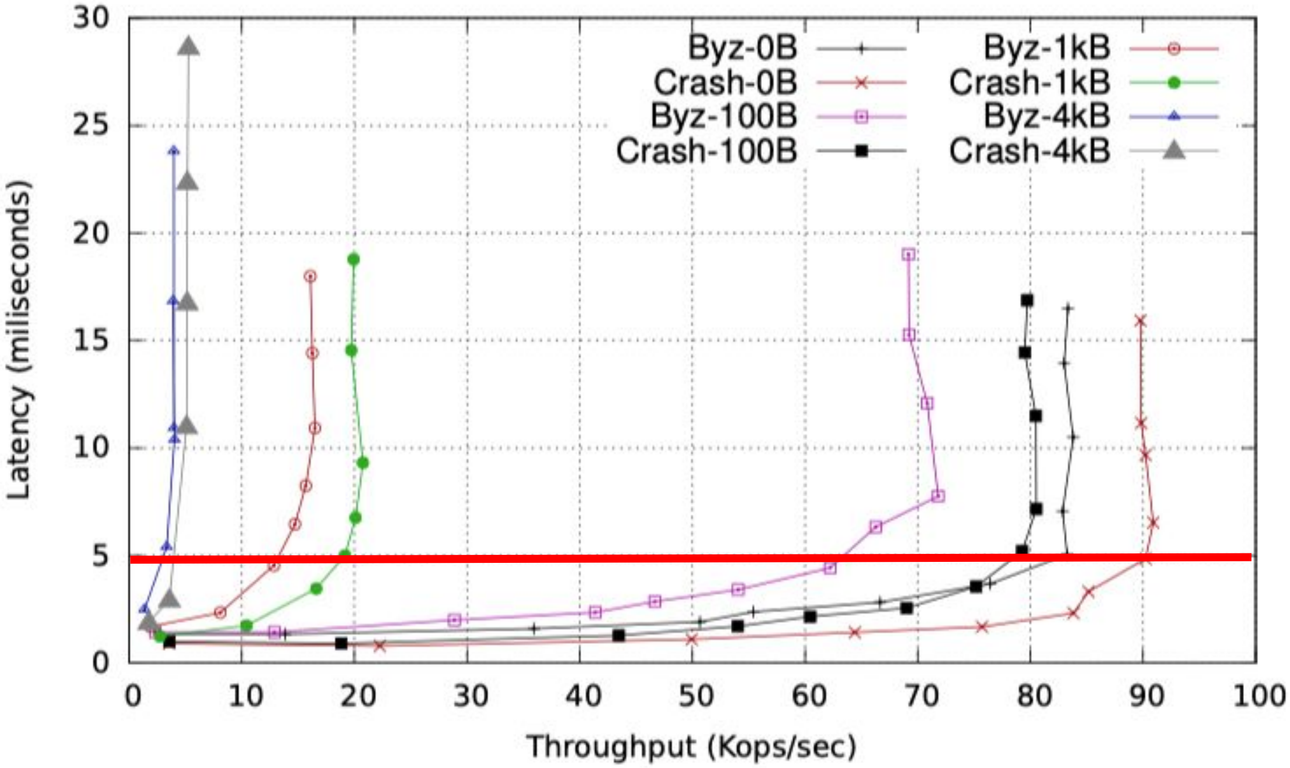


Raw Throughput and Latency

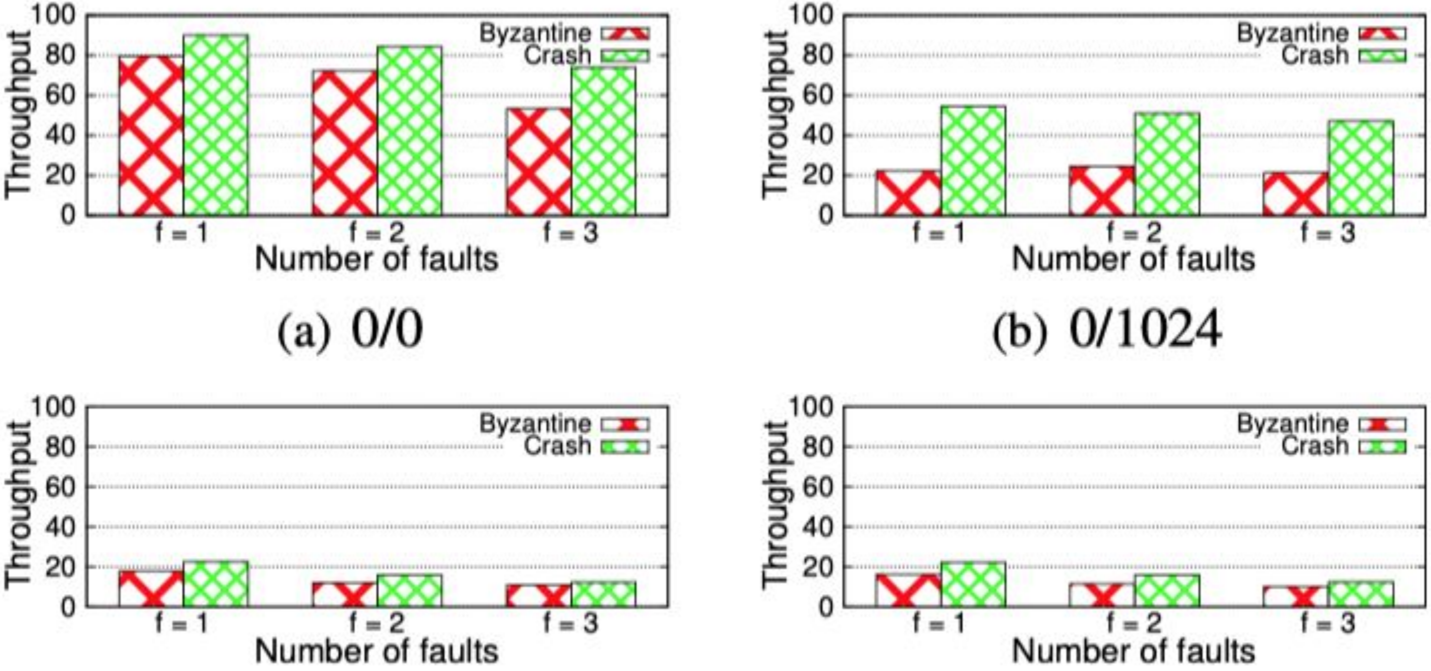
Result 1: CFT setup is always better than BFT
Result 2: Payload size increases -> BFT-SMART performance decreases



Raw Throughput and Latency



Performance in Different System



(a) 0/0

(b) 0/1024

(c) 1024/0

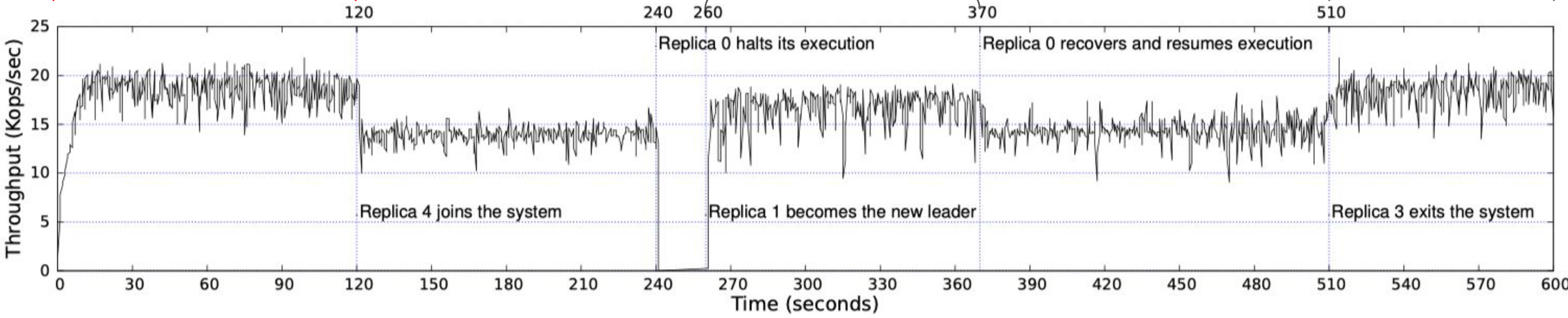
(d) 1024/1024

Performance of BFT-SMART-based System

Replica 0~3

Replica 1 becomes new leader

Replica 3 exits



Replica 0~4

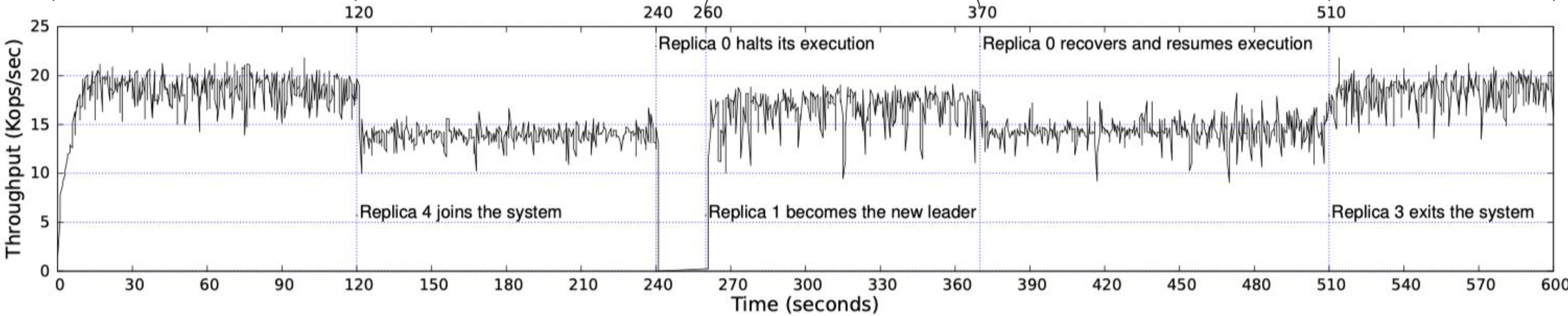
Replica 0 recovers

Performance of BFT-SMART-based System

Replica 0~3

Replica 1 becomes new leader

Replica 3 exits



Replica 0~4

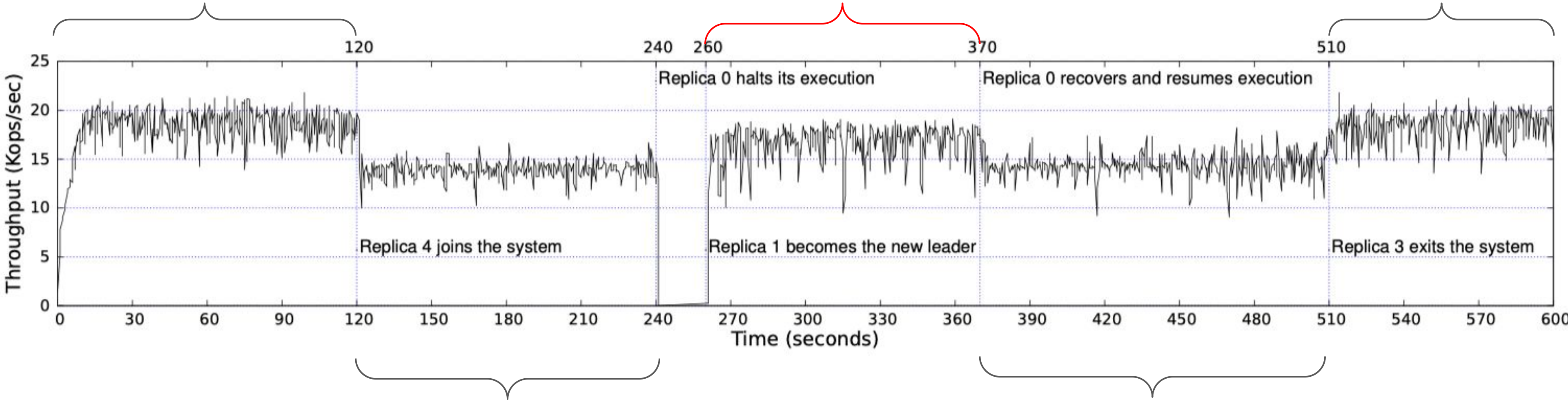
Replica 0 recovers

Performance of BFT-SMART-based System

Replica 0~3

Replica 1 becomes new leader

Replica 3 exits



Replica 0~4

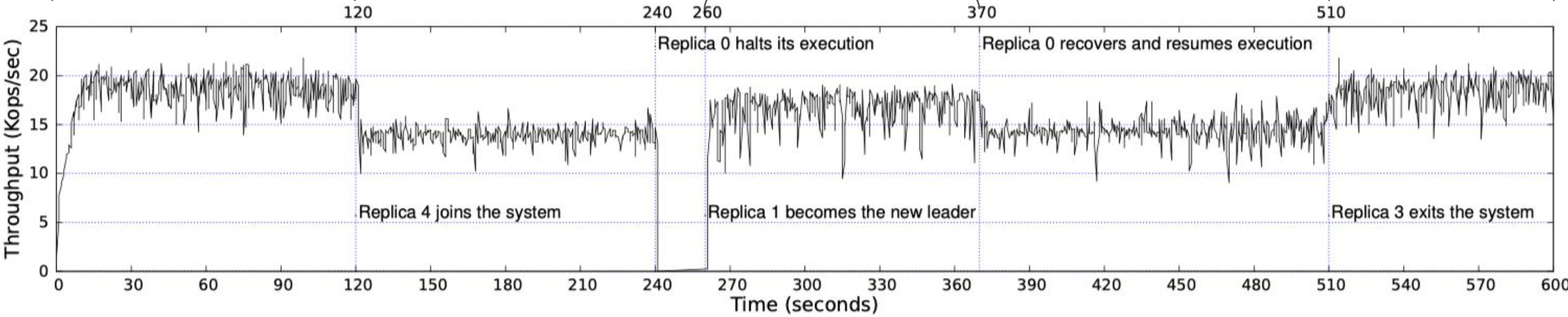
Replica 0 recovers

Performance of BFT-SMART-based System

Replica 0~3

Replica 1 becomes new leader

Replica 3 exits



Replica 0~4

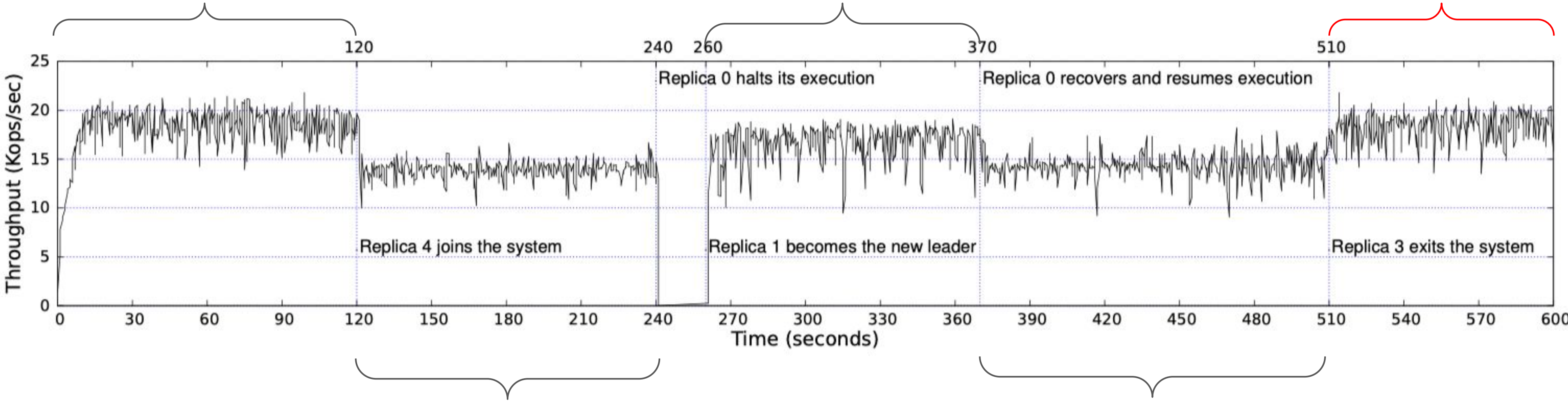
Replica 0 recovers

Performance of BFT-SMART-based System

Replica 0~3

Replica 1 becomes new leader

Replica 3 exits



Replica 0~4

Replica 0 recovers

Lessons Learned

1. BFT in Java

2. How To Test BFT

3. Dealing with Heavy Load

4. Maintenance & Robustness

Lessons Learned

1. BFT in Java

   a. Easy to use
   b. Feasible implementation of secure software

Notice: Need to be used carefully!

2. How To Test BFT

   a. Test on JUnit
   b. Identify the malicious behaviors => carefully analyze
   c. How to inject code for malicious behaviors on replicas => AOP or simple commented code

Lessons Learned

3. Dealing with Heavy Load

a. Late f replicas in message processing (cuz only needs n-f to progress)
b. non-Ordered requirements
c. Thrashing: dropping down throughput under heavy load

4. Maintenance & Robustness

a. Complex but completed

Core protocol

- Total order multicast
  - During normal execution, clients send their requests to all replicas and wait for their replies
  - Total order is achieved through consensus protocol

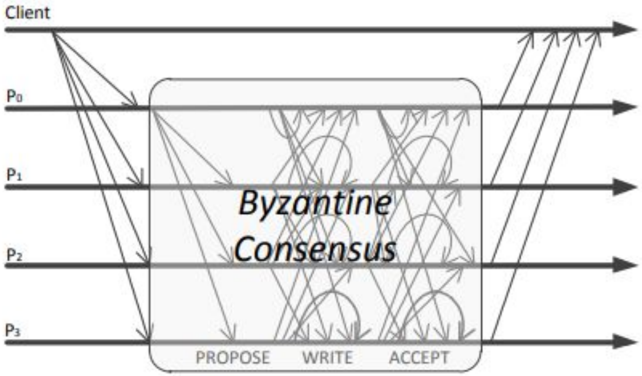


Figure 2. BFT-SMaRt normal phase message pattern.

Lessons Learned

3. Dealing with Heavy Load

   a. Late f replicas in message processing (cuz only needs n-f to progress)
   b. non-Ordered requirements
   c. Thrashing: dropping down throughput under heavy load

4. Maintenance & Robustness

   a. Complex but completed

Conclusions

1. This paper mainly report the process and results in building BFT-SMART library.

2. Describing how to implement the protocol in a safe and efficient way.

Thanks for Listening