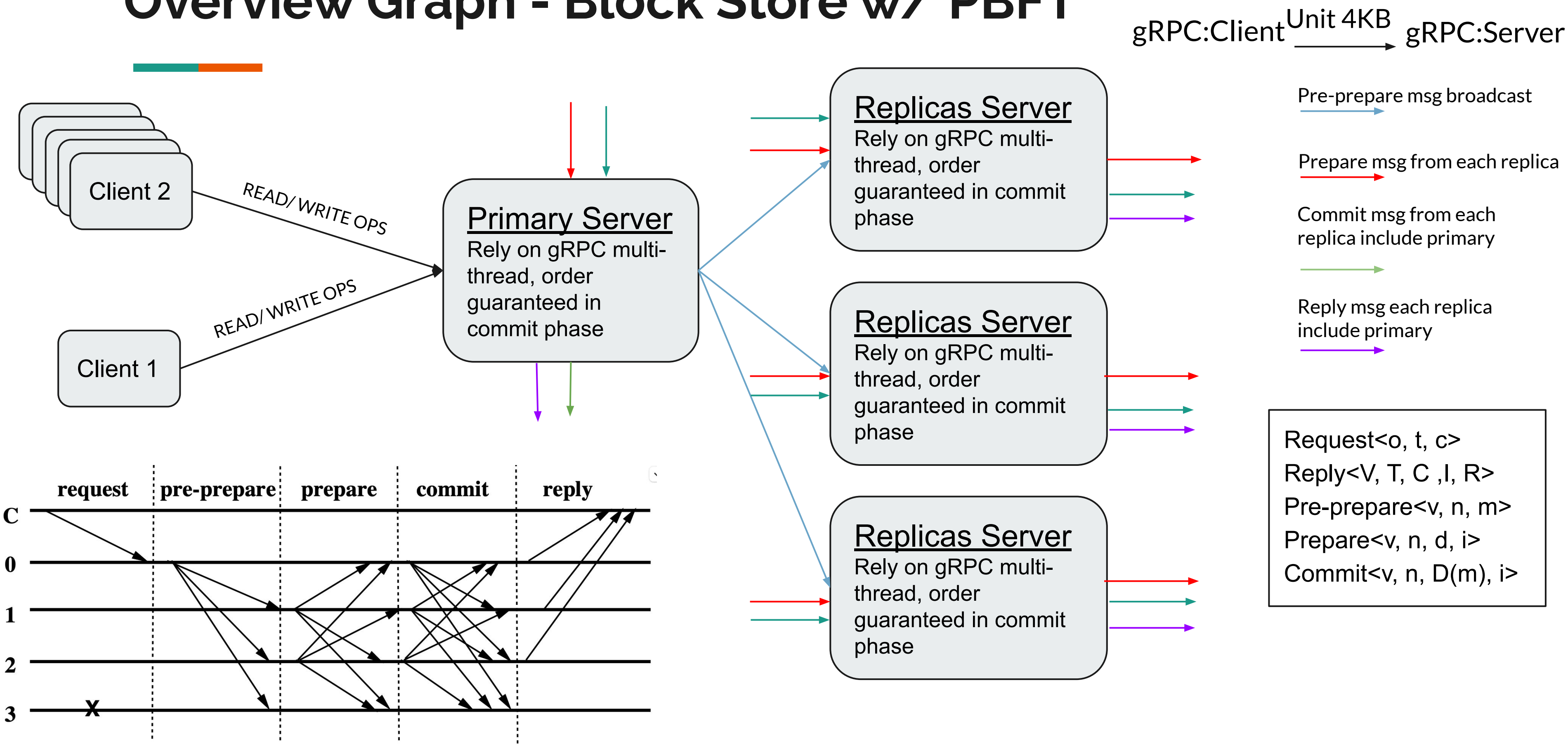
P4: Replicated Block Store w/ PBFT

Team:

Domen Su, Hao-Yu Shih, Shutao Wang, Yatharth Bindal

Overview Graph - Block Store w/ PBFT

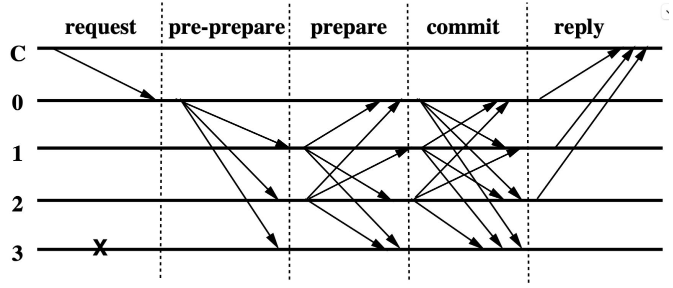


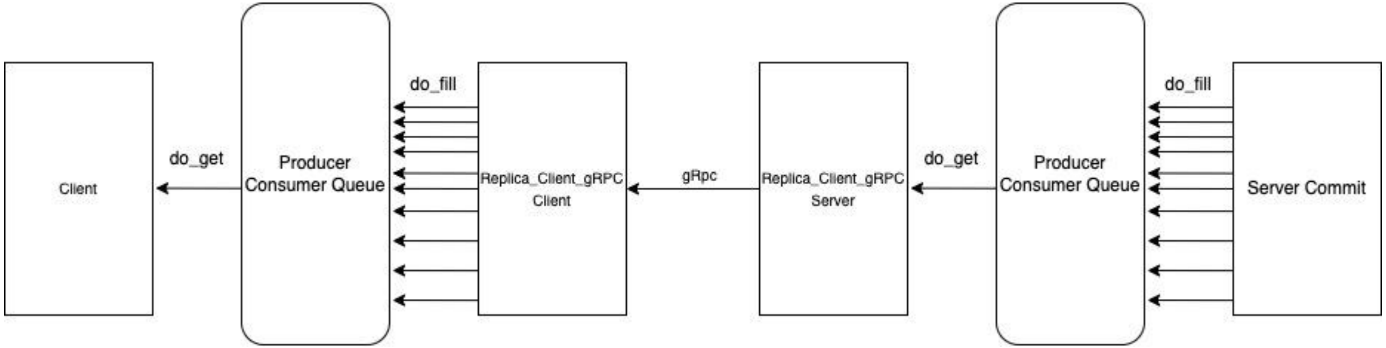
Design Decisions

- OpenSSL - SHA256, Digital Signature
- Memory cached all client operations in case of replica recovery
- Predefined Replica ID
- Client ID: as public key, replicated stored on client connection
- Does not provide view change, nor separate checkpoint mechanism
- Multithread with 3 phases to increase the performance
    - Use block on the commit phase to create the total order
    - Producer Consumer Queue

Producer Consumer Queue

- Utilize gRPC server streaming to notify result from replica to client
  - gRPC does not provide bi-directional communication natively
  - gRPC infinit running writer on server side, reader on client side
- To better utilize gRPC multi-threaded nature to have efficiency design
- Producer and consumer thread can work on different speed





Digital Signature

- **gRPC authentication**
  - Already part of gRPC easy to use
  - Not flexible enough → We need to relay & store the signed messages.

- **OpenSSL - RSA**
  - Flexible
  - We need to sign & verify the messages by ourselves.
  - The gRPC interface will become obscure
    - using generic “SignedMessage” as an argument

Crash Recovery

1. **Treat newly recovery replica as a client, send request to sync data with PBFT protocol**
    - Advantage
        - Correctness & safety - strong consistency
        - Easy to implement
    - Disadvantage
        - Not efficient- Need to run through full PBFT protocol
2. **Contact all replicas**
    - Advantage
        - Safety - contact at least one non-faulty node
        - More efficient than 1.
    - Disadvantage
        - Complex protocol
3. **Contact any replica to sync content**
    - Advantage
        - Most efficient
    - Disadvantage
        - Require an additional RPC call
        - Safety → May not trust the contacted replica → Can be solved by providing a set (2f+1) of signatures

Crash Recovery

1. **Use Merkle Tree to compare the state**
   - Advantage
     - Efficient → O(log(storage size))
   - Disadvantage
     - Difficult to implement → It is an interactive process, more difficult to provide atomicity
2. **Use hashes of blocks to compare the state**
   - Advantage
     - Non-interactive process → Easy to provide atomicity
   - Disadvantage
     - Not as efficient as 1. → O(sqrt(storage size))
3. **Replay the history**
   - Advantage
     - Easy
     - Can recover efficiently
   - Disadvantage
     - The replicas have to remember all the operation history.

Other Design Decisions

- Predefined Replica ID
- Use client public key as client ID → reduce the server states
- Use multiple conditional variable to guarantee total order in commit phase (gRPC is multi-threaded)

Fault Cases (Primary)

Scenario	Behavior
Primary ignores the client’s message	The client’s request times out and it broadcasts the request to all the replicas
Primary modifies client’s message before pre-prepare	Replicas are not able to verify the signature for the modified message and reject it
Primary assigns an incorrect sequence number to a message	Replicas detect the inconsistency after referring to their operation history and reject the message
Primary sends a prepare message	Replicas detect the origin of the message and reject prepare messages from the primary
Primary sends different messages to different replicas	Replicas fail to reach consensus

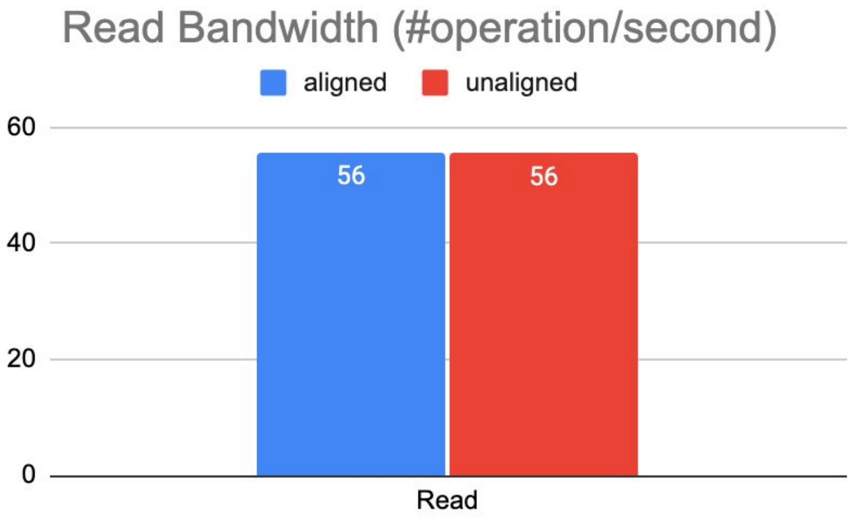
Fault Cases (Replica)

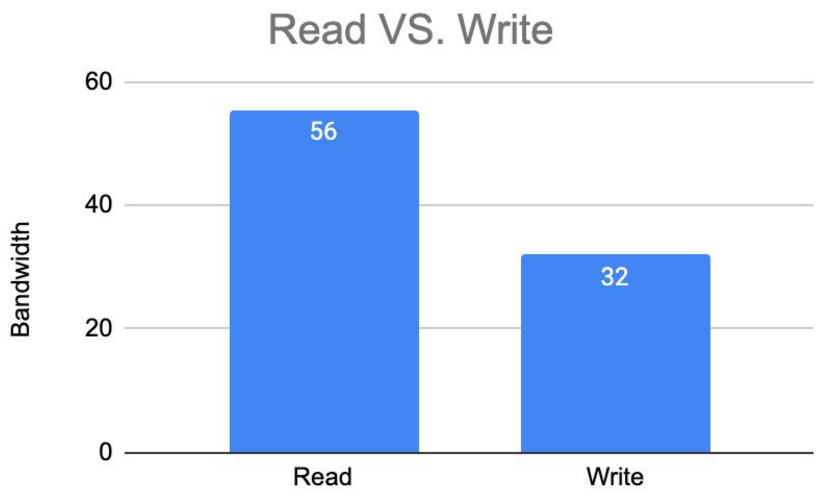
Scenario	Behavior
Replica sends a pre-prepare message	Other replicas detect it and reject the message
Replica modifies some information in the prepare stage	The modified information gets rejected by other replicas
F replicas team together and attempt to commit an incorrect message	The incorrect message fails to reach consensus as at least F+1 replicas have the correct message

Design Decisions - not implemented

- gRPC authentication
- Crash recovery
  - Treat newly recovery replica as a client to request for difference in most updated state
  - Using Checkpoint and Log to recover
- **Btrfs** for checkpoint copy-on-write
- Use **merkle tree** to reflect the state of replica, and it improve the performance of calculating checkpoints
  - Update on commit write
- Garbage Collection
  - Checkpoint
  - High watermark and low watermark

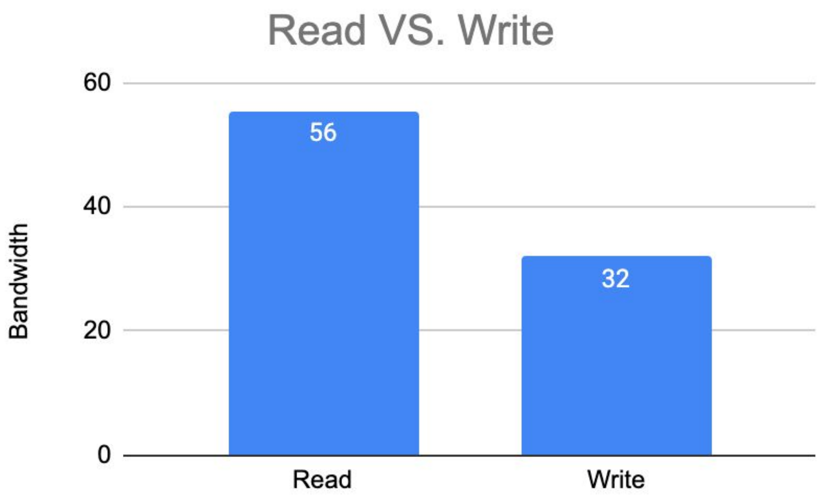
Testing Results - Performance Plots



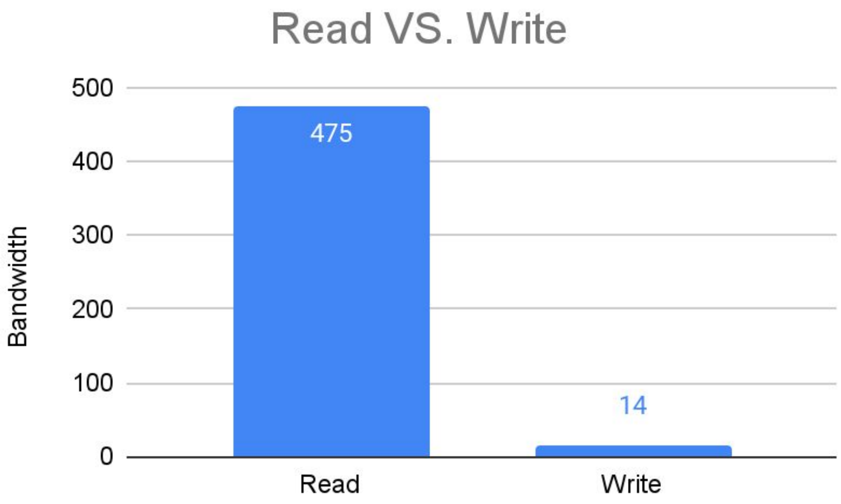


Performance - PBFT vs. Primary_Backup

PBFT

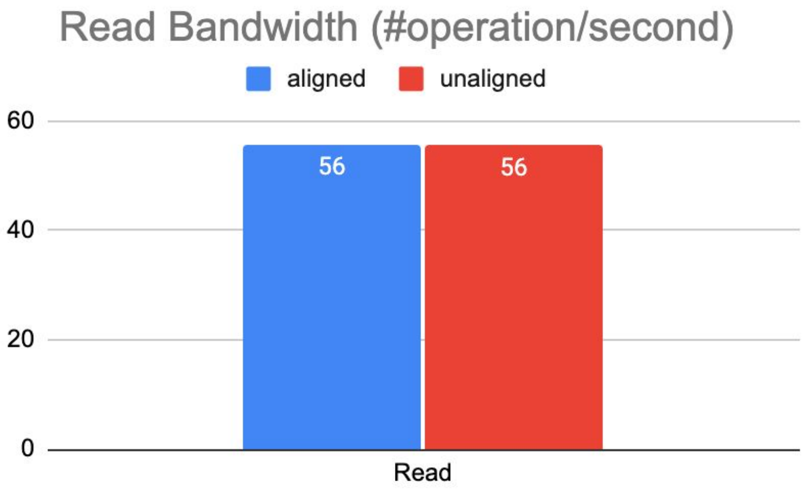


Primary_Backup

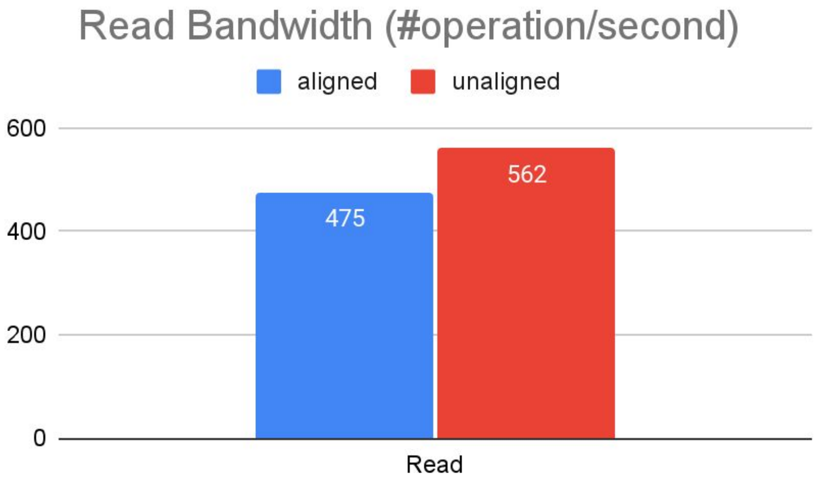


Performance - Aligned address vs unaligned address

PBFT



Primary_Backup



Thank you for listening!

Q&A