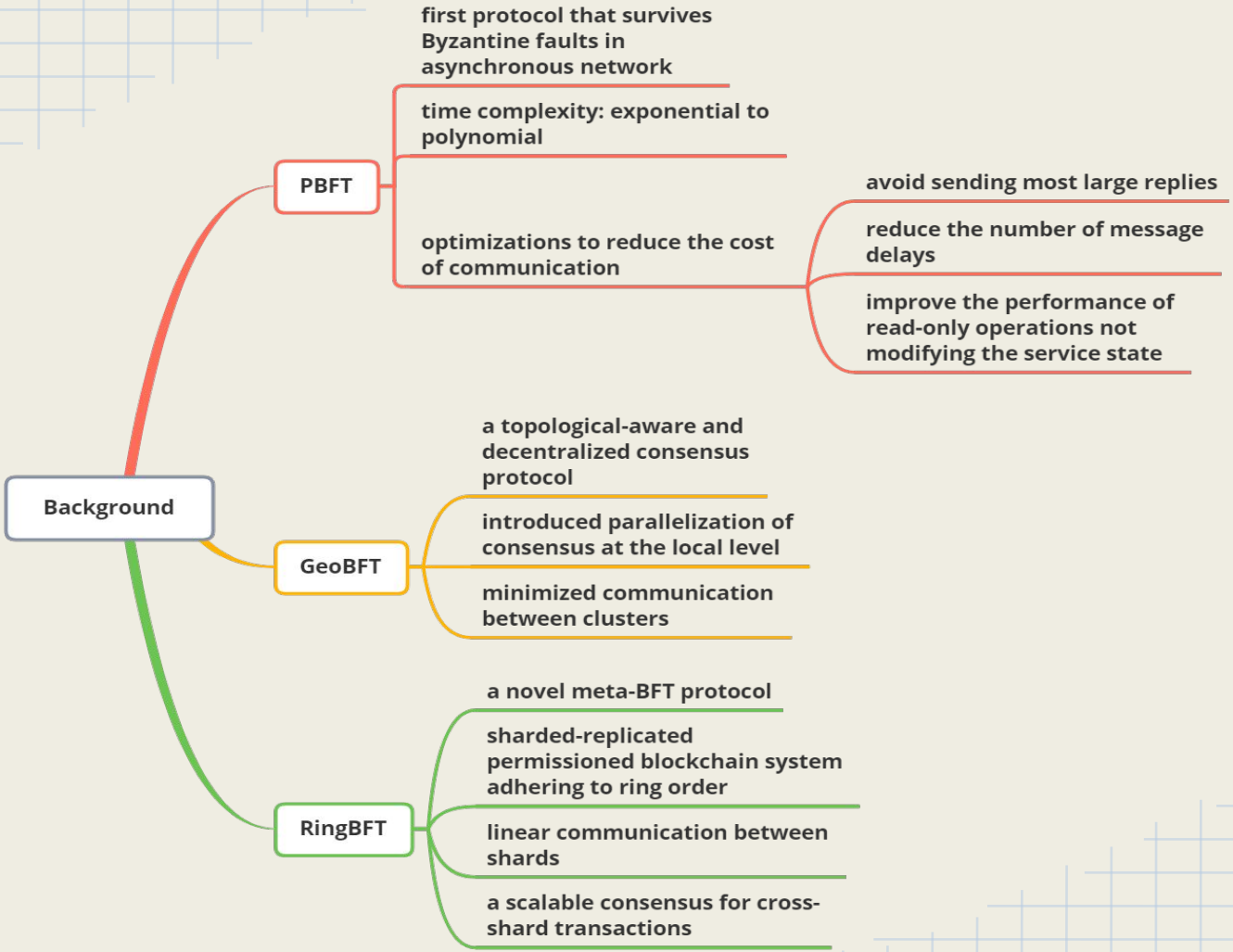
Implementation of RingBFT: Resilient Consensus over Sharded Ring Topology

Authors : Sajjad Rahnama, Suyash Gupta, Rohan Sogani, Dhruv Krishnan, Mohammad Sadoghi
Presenter: Jiangnan Chen, Fuming Fu, Haochen Yang, Weijia Wang, Xiaoxi Yu, Zaoyi Zheng
Date: Nov 29, 2021

UCDAVIS
UNIVERSITY OF CALIFORNIA

Roadmap

1. **Introduction**
   PBFT - GeoBFT - RingBFT
2. **Message Redesign in RingBFT**
3. **Message Execution in RingBFT**
4. **Global Sharing in RingBFT**


Background
PBFT
first protocol that survives Byzantine faults in asynchronous network
time complexity: exponential to polynomial
optimizations to reduce the cost of communication
avoid sending most large replies
reduce the number of message delays
improve the performance of read-only operations not modifying the service state
GeoBFT
a topological-aware and decentralized consensus protocol
introduced parallelization of consensus at the local level
minimized communication between clusters
RingBFT
a novel meta-BFT protocol
sharded-replicated permissioned blockchain system adhering to ring order
linear communication between shards
a scalable consensus for cross-shard transactions

Introduction

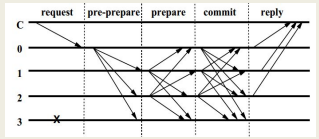
RingBFT

a solution to federated database application

Cross-shard Consensus

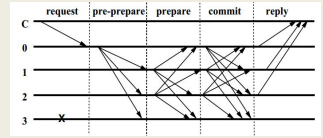
Single-shard Consensus

C1



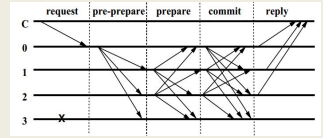
Shard 1

C2



Shard 2

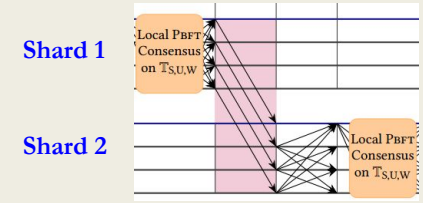
C3

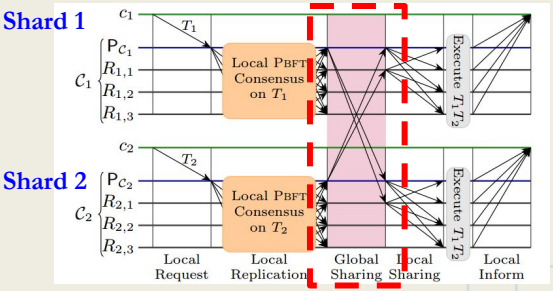


Shard 3

(1) Ring order

(3) forward process

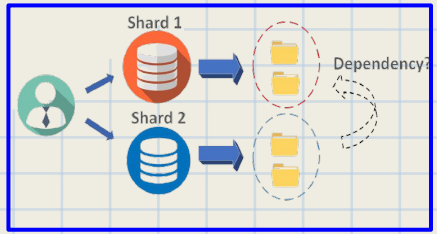




GeoBFT → RingBFT

(2) data lock

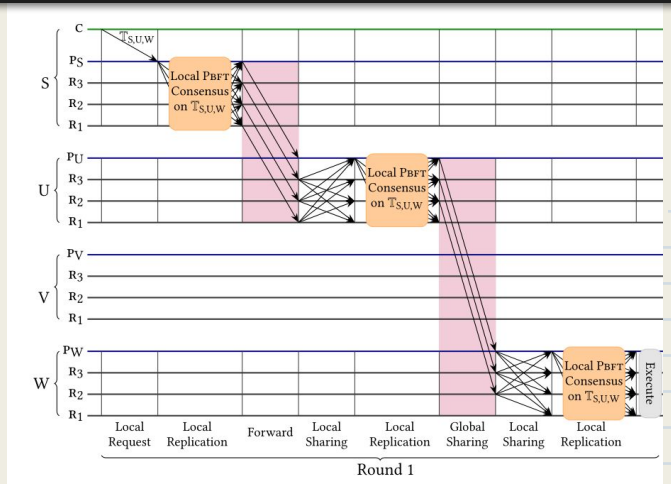
why?



Message Redesign in RingBFT

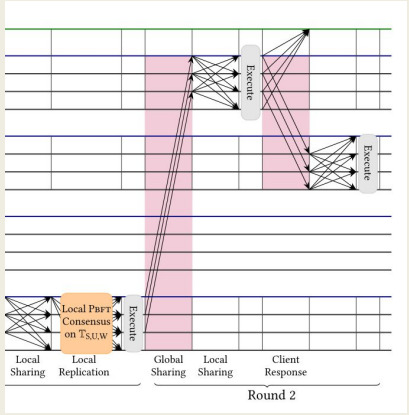
```cpp
class RingBFTForwardMessage : public Message
{
public:
    void copy_from_buf(char *buf);
    void copy_to_buf(char *buf);
    void copy_from_txn(TxnManager *txn);
    void copy_to_txn(TxnManager *txn);
    uint64_t get_size();
    void init() {}
    void release();

    void sign(uint64_t dest_node_id);
    bool validate();
    string toString();

    uint64_t view;
    Array<uint64_t> index;
    uint64_t hashSize;
    string hash;

    Array<uint64_t> signSize;
    Array<uint64_t> signOwner;
    vector<string> signatures;

    deque<uint64_t> executeOrder; // indexes of shards in execution order
    deque<uint64_t> forwardOrder; // indexes of shards in forwarding order
    // node id of next shard to forward
    uint64_t get_next_node_id(deque<uint64_t> ringOrder, deque<uint64_t> executeOrder);

}
```

```cpp
// get next node id in forwarding oder
uint64_t RingBFTForwardMessage :: get_next_node_id(deque<uint64_t> ringOrder, deque<uint64_t> executeOrder){
    uint64_t next_shard = ringOrder.front();
    ringOrder.pop_front();
    if (ringOrder.empty()){
        executeOrder.push_front(next_shard);
    }else{
        executeOrder.push_back(next_shard);
    }
    return next_shard;
}
```



Message Redesign in RingBFT

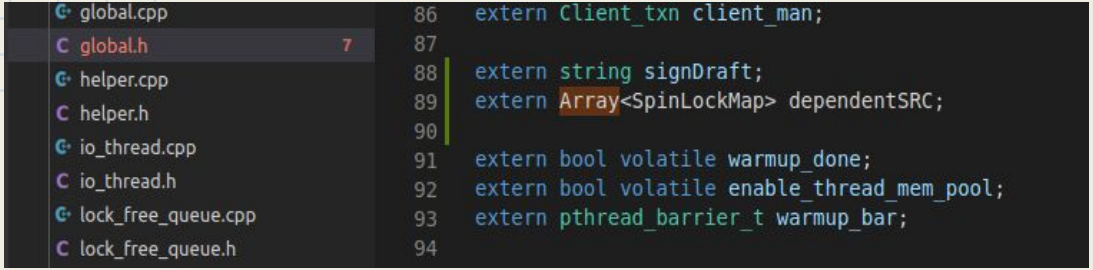
```
class RingBFTExecuteMessage : public Message
{
public:
    void copy_from_buf(char *buf);
    void copy_to_buf(char *buf);
    void copy_from_txn(TxnManager *txn);
    void copy_to_txn(TxnManager *txn);
    uint64_t get_size();
    void init() {}
    void release();

    void sign(uint64_t dest_node_id);
    bool validate();
    string toString();

    uint64_t view;
    Array<uint64_t> index;
    uint64_t hashSize;
    string hash;

    Array<uint64_t> signSize;
    Array<uint64_t> signOwner;
    vector<string> signatures;

    RingBFTExecuteMessage(deque<uint64_t> eo);
    deque<uint64_t> executeOrder; // indexes of shards in execution order

    // node id of next shard to forward
    uint64_t get_next_node_id(deque<uint64_t> executeOrder);
}
```




```
RingBFTExecuteMessage::RingBFTExecuteMessage(deque<uint64_t> eo){
    executeOrder = eo;
}
```

```
// get next node id in execution oder
uint64_t RingBFTExecuteMessage::get_next_node_id(deque<uint64_t> executeOrder){
    uint64_t next_executeion_node_id = executeOrder.front();
    executeOrder.pop_front();
    return next_executeion_node_id;
}
```

Message Execution in RingBFT



Design dependent transaction: Modified SpinLockMap to enable every node of shard to lock and unlock data fragments.

Use the array of SpinLockMap to represent the datafragment needed for every execution.

Message Execution in RingBFT

```
108     switch (msg->get_rtype())
109     {
110     case KEYEX:
111         rc = process_key_exchange(msg);
112         break;
113     case CL_BATCH:
114         rc = process_client_batch(msg);
115         break;
116     case BATCH_REQ:
117         rc = process_batch(msg);
118         break;
119     case PBFT_CHKPT_MSG:
120         rc = process_pbft_chkpt_msg(msg);
121         break;
122     case RBFT_LAST_ROUND_MSG:
123         send_last_round_execute_msg(msg);
124     case EXECUTE_MSG:
125         rc = process_execute_msg(msg);
126         break;
```

WorkThread:

1. Get message from workQueue
2. Process the message with regard to its type
3. Process first round execution and last round execution separately

```
804 void WorkerThread::send_last_round_execute_msg(Message *msg)
805 {
806     rbft_msg = (RingBFTExecuteMessage *)msg
807     vector<uint64_t> dest;
808     dest.push_back(rbft_msg->executeOrder.front());
809     RingBFTExecuteMessage lastMsg = Message::create_message(RBFT_LAST_ROUND_MSG);
810     msg_queue.enqueue(get_thd_id(), lastMsg, dest);
811     dest.clear();
812 }
```

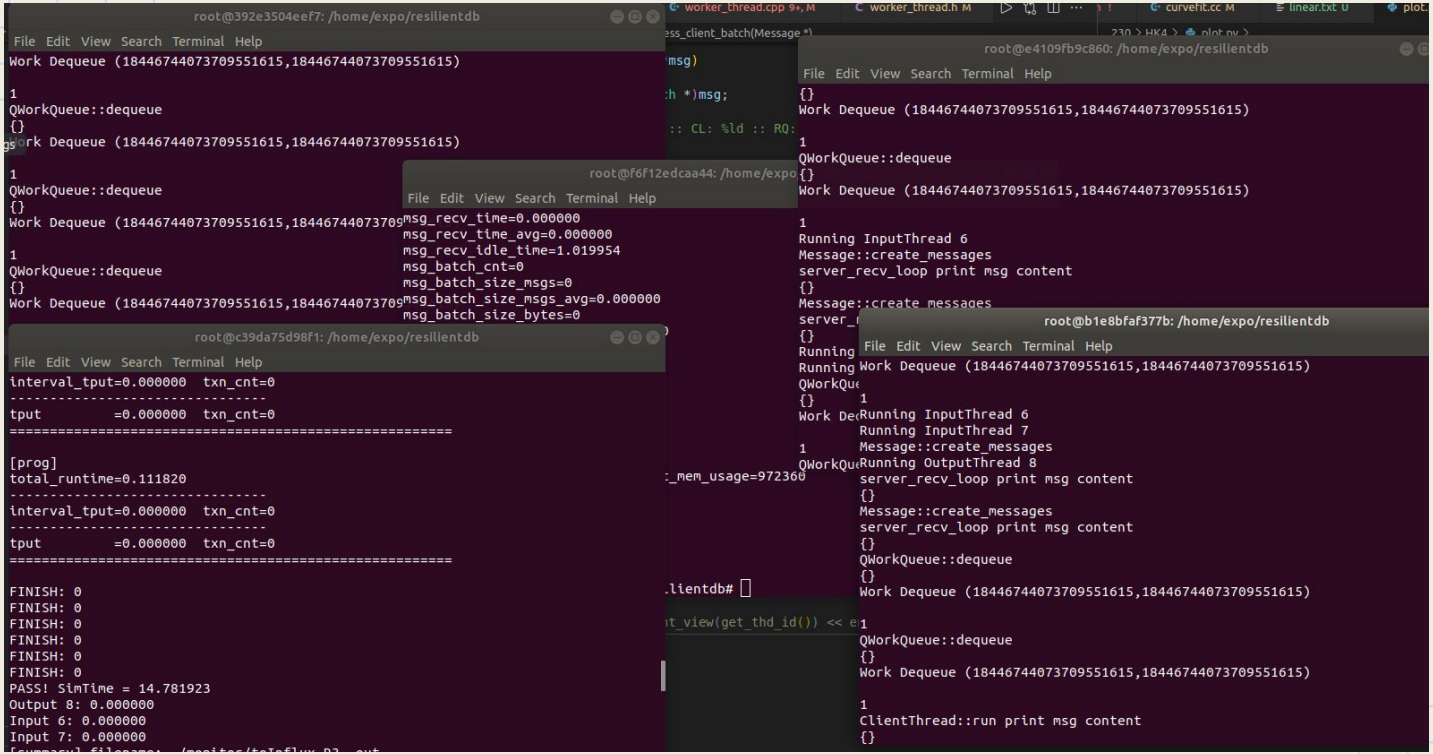
Message Execution in RingBFT

```
845     RingBFTExecuteMessage rbft_msg = (RingBFTExecuteMessage *)msg;
846     if (msg->rtype != RBFT_LAST_ROUND_MSG)
847     {
848         //lock corresponding datafragment
849         dependentSRC[rbft_msg->executeOrder.front()].lock();
850         //If the last shard in transaction-involved shards
851         if (rbft_msg->executeOrder.size() == 1)
852         {
853             //The id of initiator is set to 0
854             vector<uint64_t> dest;
855             dest.push_back((uint64_t)0);
856             lastMsg = create_message(RBFT_LAST_ROUND_MSG);
857             msg_queue.enqueue(get_thd_id(), lastMsg, dest);
858             dest.clear();
859         }
860     }
861     else
862     {
863         //execute
864         tman->run_txn_print(msg);
865         //reply to client
866         Message *rsp = Message::create_message(CL_RSP);
867         ClientResponseMessage *crsp = (ClientResponseMessage *)rsp;
868         crsp->init();
869         crsp->copy_from_txn(txn_man);
870         vector<uint64_t> dest;
871         dest.push_back(txn_man->client_id);
872         msg_queue.enqueue(get_thd_id(), crsp, dest);
873         dest.clear();
874         //unlock corresponding datafragment
875         dependentSRC[rbft_msg->executeOrder.front()].unlock();
876     }
877     return RCOK;
```

First Round Execution:
1.lock the data fragment
2.check if the last one

Last Round Execution:
1.perform the transaction
2.unlock corresponding data fragment
3.reply to client

Message Execution in RingBFT



Execution:
Message cannot be displayed

Global Sharing in RingBFT

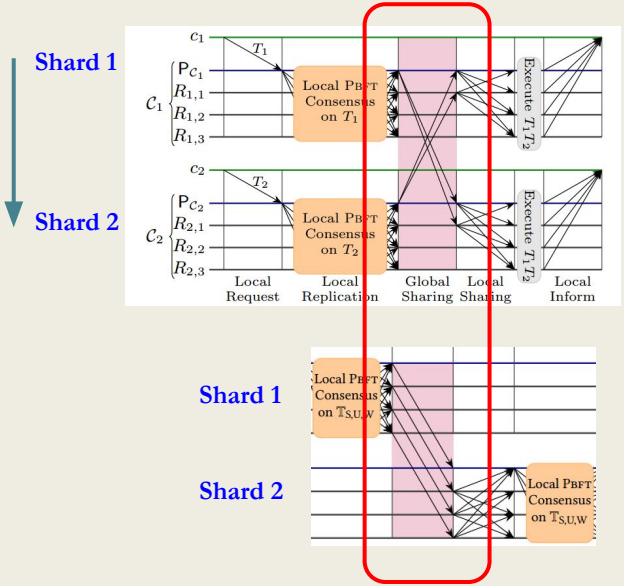
Difference:

GeoBFT: Primary -> Replica of other shard, send f+1 messages

RingBFT: Both replica and primary send one message to next shard which has the same id

GeoBFT

Ring order



RingBFT

Global Sharing in RingBFT

Difference:

GeoBFT: Primary -> Replica of other shard, send f+1 messages

RingBFT: Both replica and primary send one message to next shard which has the same id

```
#if RING
        // Forward message(Global sharing in RingBFT)
        if (g_node_cnt > ringbft_cluster_size) // Not for Single shard
        {
            RingBFTForwardMessage *rbm = (RingBFTForwardMessage *)Message::create_message(txn_man, RINGBFT_MSG);
            rbm->txn_id = txn_man->get_txn_id();

            vector<uint64_t> dest;
            for (uint64_t i = 0; i < g_node_cnt; i++)
            {
                // Not in the same shard and having the same id
                if (!is_in_same_cluster(g_node_id, i) && i % ringbft_cluster_size == g_node_id % ringbft_cluster_size)
                {
                    dest.push_back(i);
                    break;
                }
            }

            msg_queue.enqueue(get_thd_id(), rbm, dest);
            dest.clear();
        }
#endif
```

- For cross-shard, number of nodes should be more than that in a cluster
- Create message and coerce it
- Register in transaction manager
- Traverse all nodes:
  1. Not in the same cluster
  2. Have the same id
  3. In adjacent shards(Ring order)
- Add to message queue then clear the destination vector

The End

UCDAVIS
UNIVERSITY OF CALIFORNIA