# Hyperledger Fabric

Introduction

# Outline

1. Introduction of PBFT
2. Mechanism of PBFT
3. Proof of PBFT
4. Detail of PBFT
5. Optimization of PBFT
6. Application: Hyperledger
7. Microsoft talking time

# Introduction

•••

# What is PBFT

1. Practical Byzantine Fault Tolerance
    a. A consensus algorithm
        i. solving Byzantine Fault Tolerance
        ii. versus to proof of work
    b. Replication System
        i. Not just used for Blockchain, it can be seen as NFS
        (Network file system)

# PBFT vs POW

1. Node Scalability
    a. POW (no limited) > PBFT (4~100 node)
2. Fault Tolerance
    a. POW(50%) > PBFT (25~33%)
3. Transaction Throughput
    a. PBFT(1000/s  Fabirc) >> POW(7/s  Bitcoin)
    b. Make it practical !!!

# PBFT Properties

1. No need mining
2. Fault tolerance :  $n = 3f + 1$
3. Every node must connect with all nodes
4. View  concept
    a. In a view, one of the nodes is the primary node.
    b. Others will be replica node.

# Mechanism

...

# What is PBFT

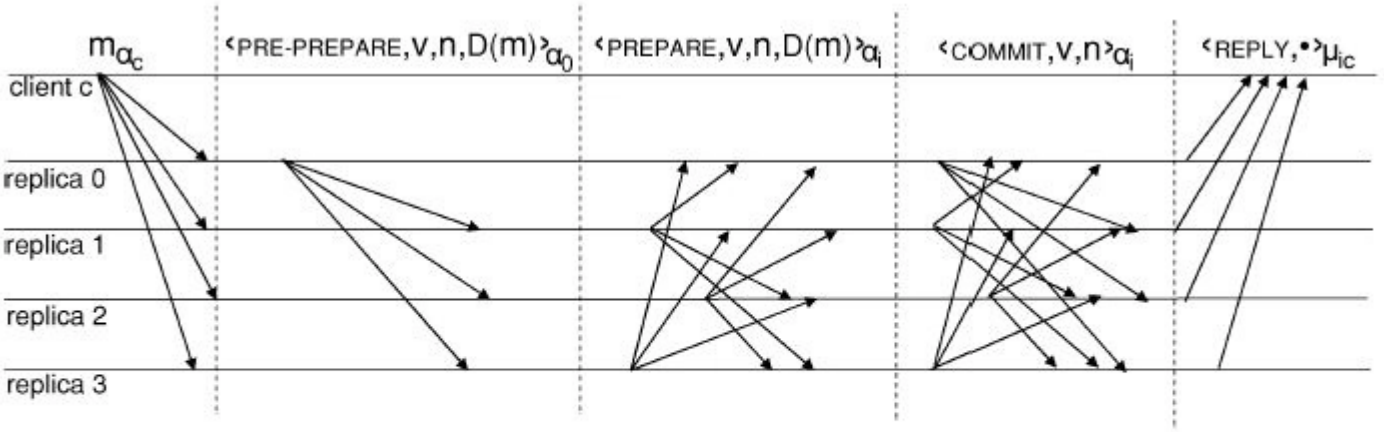
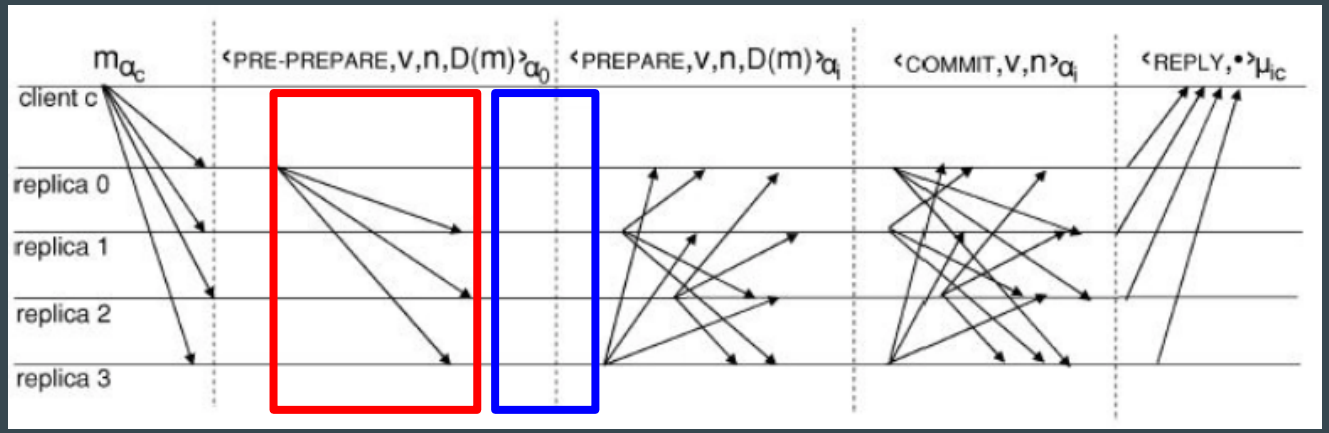



Fig. 1. Normal case operation: the primary (replica 0) assigns sequence number $n$ to request $m$ in its current view $v$ and multicasts a PRE-PREPARE message with the assignment. If a backup agrees with the assignment, it multicasts a matching PREPARE message. When a replica receives messages that agree with the assignment from a quorum, it sends a COMMIT message. Replicas execute $m$ after receiving COMMIT messages from a quorum.

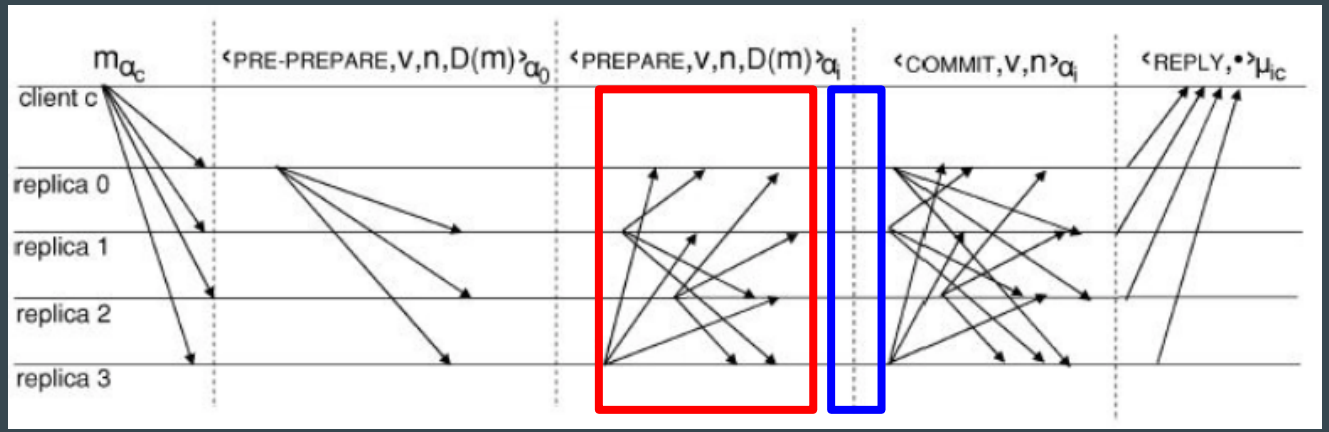
# What is PBFT




Pre-Prepare :

1. Primary node validate & Multicast the request got from client
2. Validate & Order requests got from primary node.
    a. Distinguish the primary node is bad node when the digest of requests is different between primary node and replica node.

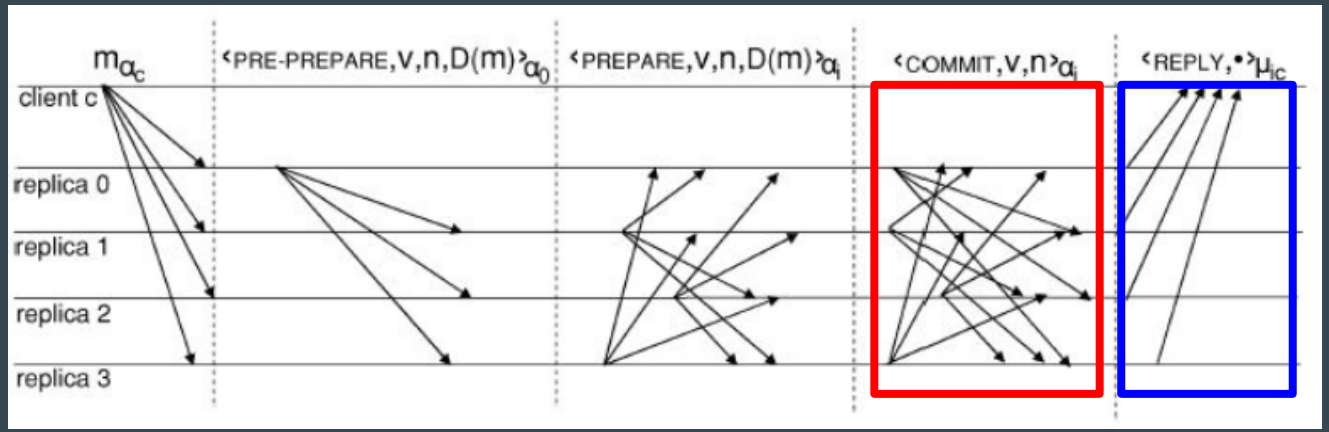
# What is PBFT




Prepare :

1. Multicast the requests the node got and compare from other node. Make sure all the node got the same requests at same time.
2. Execute these requests

# What is PBFT




Commit:

1. Multicast to other node the node have executed, then compare with other node's state. Make sure every node have the same current state.
2. Each node write in NFS and replies conclusion to client.

# Proof of PBFT

...

# Define a good node

1. Always processing request correctly
    a. Deterministic
2. Time limited Network
    a. the Interval sending message and getting reply
        always
        limited
3. Otherwise, the node is seen as bad node

# Many kinds of attack

1. Control the node
    a. Tempering the message
2. Delay the Network
    a. Denial of service
3. Pretend as good node

# Tempering the message

1. Voting

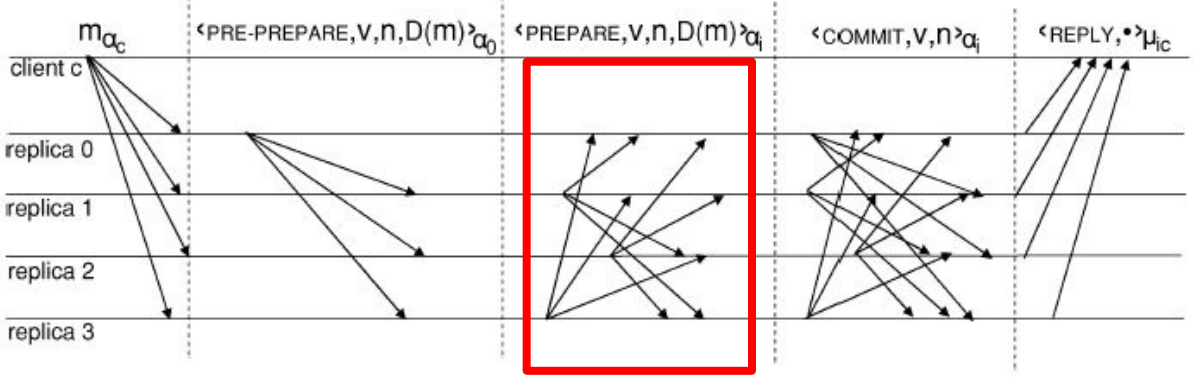



Fig. 1. Normal case operation: the primary (replica 0) assigns sequence number $n$ to request $m$ in its current view $v$ and multicasts a PRE-PREPARE message with the assignment. If a backup agrees with the assignment, it multicasts a matching PREPARE message. When a replica receives messages that agree with the assignment from a quorum, it sends a COMMIT message. Replicas execute $m$ after receiving COMMIT messages from a quorum.

# Delay the network

1. Once receive 2f + 1 of reply
    a. Make the decision and keep process.
2. All the bad node give bad reply
    a. Can not change the decision.
3. All the bad node delay to replay
    a. Can not delay the decision.

# Pretend as good node

1. Change private key periodically
2. Turning to recover node detect which the node loss the control.
3. Other node reject message sign by old key.

# Something detail

...

# View Change

1. If the primary node is bad node

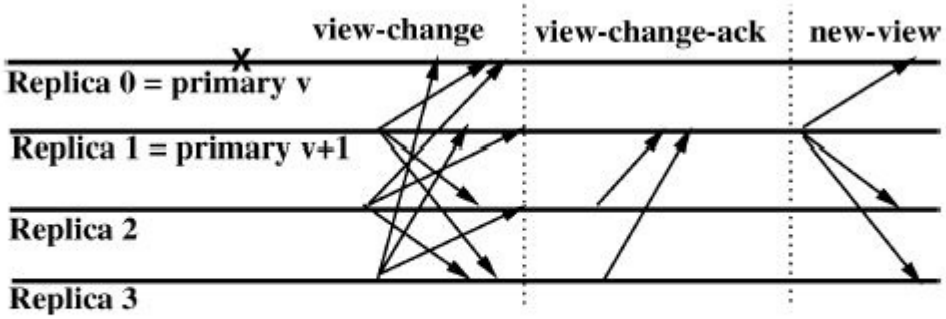



Fig. 2. View-change protocol: the primary for view $v$ (replica 0) fails causing a view change to view $v+1$.

# Garbage collection

1. Any message received or sent to other node will log in nodes.
2. The log will growth infinitly.
3. Use checkpoint consider which state is absolutly correct.
4. Then discard the logs recording prior to that state
5. Once the k requests committed, multicast checkpoint message
6. Get 2f+1 replay, then discard the logs

# Optimalize

...

# Chunk Processing

1. Seen as numbers of request a chunk.
2. Sending message in unit of chunk.
3. Decrease network usage, amortize protocol overhead.

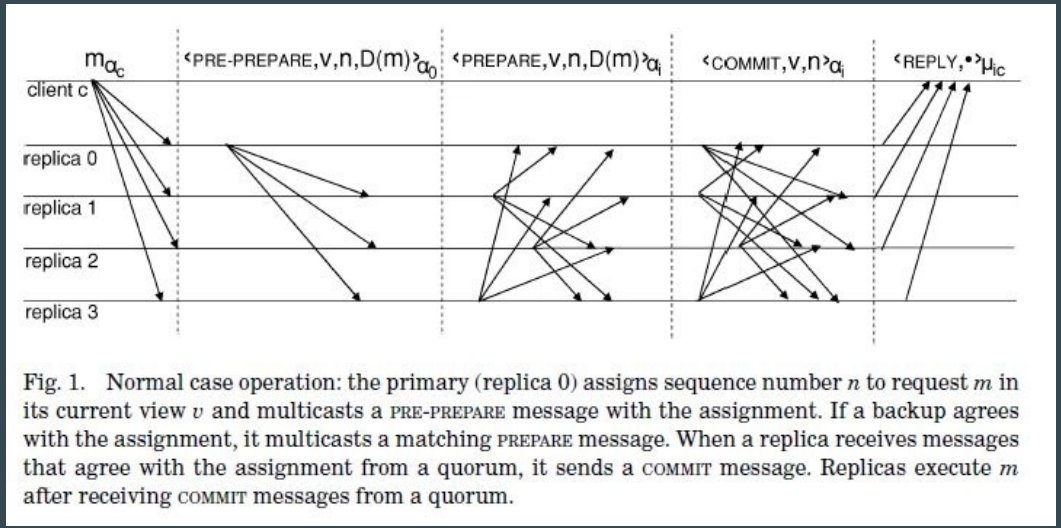



Fig. 1. Normal case operation: the primary (replica 0) assigns sequence number $n$ to request $m$ in its current view $v$ and multicasts a PRE-PREPARE message with the assignment. If a backup agrees with the assignment, it multicasts a matching PREPARE message. When a replica receives messages that agree with the assignment from a quorum, it sends a COMMIT message. Replicas execute $m$ after receiving COMMIT messages from a quorum.

# Digest

1. Instead of sending whole requests content
2. Just send the encode of requests
3. Decrease network usage
4. MD5

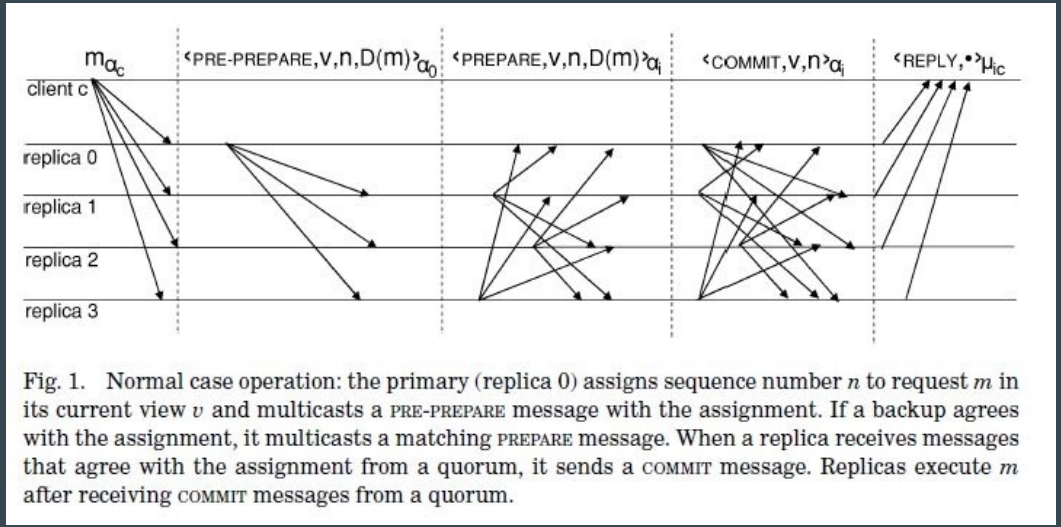



Fig. 1. Normal case operation: the primary (replica 0) assigns sequence number $n$ to request $m$ in its current view $v$ and multicasts a PRE-PREPARE message with the assignment. If a backup agrees with the assignment, it multicasts a matching PREPARE message. When a replica receives messages that agree with the assignment from a quorum, it sends a COMMIT message. Replicas execute $m$ after receiving COMMIT messages from a quorum.

# Others

1. Using symmetric cryptography authenticate message.
    a. Every node node have private key, use non symmetric cryptography exchange public key.
2. According to paper, the speed of BFT NFS just faster than normal NFS 2% to slower it 24%

Application

•••

Hyperledger Fabric

# Hyperledger Fabric

1. Website
2. Using PBFT as consensus algorithm
3. Using Nodejs / Java as client interface
4. Using golang as chaincode smart contract
    a. No cryptocurrency, UTXO, just a ledger you can record everything.
    b. No fee for transaction

# Hyperledger Fabric

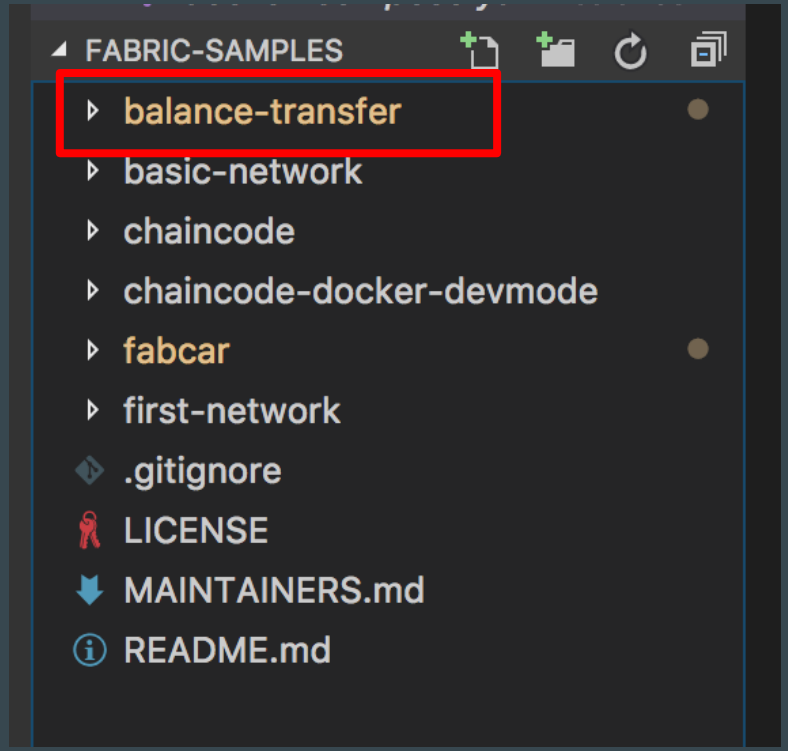
1. Require & version
    a. docker 17.03.0-ce or greater
    b. docker-compose 1.8 or greater
    c. Go language 1.7.x
    d. Nodejs 6.9.x or greater but < 7.x
    e. npm 3.10.10

# Hyperledger Fabric

1. Download the docker Image
    a. sudo curl -sSL https://goo.gl/5ftp2f | bash
    b. sudo docker images

2. Download the example
    a. git clone https://github.com/hyperledger/fabric-samples.git
    b. cd fabric-samples

# Hyperledger Fabric



# Hyperledger Fabric

1. sudo bash runApp.sh
2. run sudo docker ps see im

```
NAMES
peer1.org2.example.com
peer1.org1.example.com
peer0.org1.example.com
peer0.org2.example.com
ca_peerOrg1
orderer.example.com
ca_peerOrg2
```

# Hyperledger Fabric

1. See testAPI.sh to know the node client rest api format.
2. artifacts/src/github.com/example.cc folder
    a. put the example code of chaincode smart contract.
3. app.js show how the node rest api working
4. ChainCode package shim