Modern Large-Scale Data Management Systems after 40 Years of Consensus

Mohammad Javad Amiri, Divyakant Agrawal, Amr El Abbadi
Reaching Agreement in the Presence of Faults

M. PEASE, R. SHOSTAK, AND L. LAMPORT

SRJ International, Menlo Park, California

ABSTRACT. The problem addressed here concerns a set of isolated processors, some unknown subset of which may be faulty, that communicate only by means of two-party messages. Each nonfaulty processor has a private value of information that must be communicated to each other nonfaulty processor. Nonfaulty processors always communicate honestly, whereas faulty processors may lie. The problem is to devise an algorithm in which processors communicate their own values and relay values received from others that allows each nonfaulty processor to infer a value for each other processor. The value inferred for a nonfaulty processor must be that processor’s private value, and the value inferred for a faulty one must be consistent with the corresponding value inferred by each other nonfaulty processor.

It is shown that the problem is solvable for, and only for, $n \geq 3m + 1$, where $m$ is the number of faulty processors and $n$ is the total number. It is also shown that if faulty processors can refuse to pass on information but cannot falsely relay information, the problem is solvable for arbitrary $n \geq m \geq 0$. This weaker assumption can be approximated in practice using cryptographic methods.

KEY WORDS AND PHRASES. agreement, authentication, consistency, distributed executive, fault avoidance, fault tolerance, synchronization, voting

CR CATEGORIES: 3.81, 4.39, 5.29, 5.39, 6.22

© 1980 ACM 0004-5411/80/0400-0228 $00 75
Fault Tolerance

• Build systems that tolerate machine and network faults
• Replicate data on multiple servers to enhance availability
  • Uses State Machine Replication
  • Needs to ensure that replicas remain consistent
  • Needs consensus among different servers
Fault Tolerance

• Build systems that tolerate machine and network faults
• Replicate data on multiple servers to enhance availability
  • Uses State Machine Replication
  • Needs to ensure that replicas remain consistent
  • Needs consensus among different servers
Fault Tolerance

• Build systems that tolerate machine and network faults
• Replicate data on multiple servers to enhance availability
  • Uses State Machine Replication
  • Needs to ensure that replicas remain consistent
  • Needs consensus among different servers
Fault Tolerance

• Build systems that tolerate machine and network faults
• Replicate data on multiple servers to enhance availability
  • Uses State Machine Replication
  • Needs to ensure that replicas remain consistent
  • Needs consensus among different servers
Fault Tolerance

• Build systems that tolerate machine and network faults
• Replicate data on multiple servers to enhance availability
  • Uses State Machine Replication
  • Needs to ensure that replicas remain consistent
  • Needs consensus among different servers
Fault Tolerance

• Build systems that tolerate machine and network faults
• Replicate data on multiple servers to enhance availability
  • Uses State Machine Replication
  • Needs to ensure that replicas remain consistent
  • Needs consensus among different servers
Fault Tolerance

• Build systems that tolerate machine and network faults
• Replicate data on multiple servers to enhance availability
  • Uses State Machine Replication
  • Needs to ensure that replicas remain consistent
  • Needs consensus among different servers
A set of distributed nodes need to reach agreement on a single value
Google Bigtable
Google Bigtable

Distributed lock service
Google Spanner

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Amazon DynamoDB

SCALABILITY

- Table: Partitions
  - Reading: $0.001/MB
  - Writing: $0.01/MB
- Reserved Capacity
- Provisioned Capacity
- Data Divided Equally
- Before Split
  - 1/3
  - 1/3
  - 1/3
- After Split
  - A
  - B
  - C

PRICING

- Read Capacity Unit
- Write Capacity Unit
- 1 Read per second, up to 1 MB
- 1 Write per second, up to 1 MB

ESSENTIALS

- Key-Value Document Store
- Partitions Hold up to 10 GB
- Items: $0.014/10K
- Table: $0.11/GB

SECURITY

- AWS Identity
- AWS CloudTrail
- AWS Config
- AWS WAF

AVAILABILITY

- DynamoDB
- Multiregion
- GlobalTables
- Server-Side Encryption
- Server-Side Snapshots
- High Speed EBS Volumes

DynamoDB - a fast, flexible NoSQL database
Hyperledger Fabric (Permissioned Blockchain)
Hyperledger Fabric (Permissioned Blockchain)

IBM

Hyperledger

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Bitcoin (Permissionless Blockchain)

Blockchain data structure

Block 0 (Genesis Block)
- 50 BTC
- Transaction D
- Transaction E

Block n-1
- Transaction G
- Transaction H

Block n

Peer-to-peer overlay network

Miner

Replication

Proof-of-Work

Wallet

Block Propagation

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Bitcoin (Permissionless Blockchain)
State Machine Replication

Clients
State Machine Replication

Consensus Module

State Machine

Log

add jmp mov shl

Consensus Module

State Machine

Log

add jmp mov shl

Consensus Module

State Machine

Log

add jmp mov shl

Clients

Servers

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
State Machine Replication

- Replicated log: *replicated state machine*
  - All servers execute same commands in the same order
  - Commands are deterministic
- Consensus module ensures proper log replication
State Machine Replication

- Replicated log: replicated state machine
  - All servers execute same commands in the same order
  - Commands are deterministic
- Consensus module ensures proper log replication
State Machine Replication

- Replicated log: replicated state machine
  - All servers execute same commands in the same order
  - Commands are deterministic
- Consensus module ensures proper log replication
State Machine Replication

- Replicated log: replicated state machine
  - All servers execute same commands in the same order
  - Commands are deterministic
- Consensus module ensures proper log replication

Clients

Servers
State Machine Replication

- Replicated log: replicated state machine
  - All servers execute same commands in the same order
  - Commands are deterministic
- Consensus module ensures proper log replication
State Machine Replication

- Replicated log: replicated state machine
  - All servers execute same commands in the same order
  - Commands are deterministic
- Consensus module ensures proper log replication
Properties

Safety (bad things never happen)

- Only a value that has been proposed may be chosen
- Only a single value is chosen
- A node never learns that a value has been chosen unless it has been
Properties

Safety (bad things never happen)
- Only a value that has been proposed may be chosen
- Only a single value is chosen
- A node never learns that a value has been chosen unless it has been

Liveness (good things eventually happen)
- Some proposed value is eventually chosen
- If a value has been chosen, a node can eventually learn the value
Synchrony Mode

1. Consensus Protocols

2. Failure Model

3. Processing Strategy

4. Participants Awareness

5. Complexity Metrics
First Aspect: Synchrony Mode

Synchronous System

- Assume known bounds on message delays and process speeds
- All communication proceeds in rounds.
- In one round, a process may send all the messages it requires, while receiving all messages from others
- No message from one round may influence any messages sent within the same round.
First Aspect: Synchrony Mode

Synchronous System
- Assume known bounds on message delays and process speeds
- All communication proceeds in rounds.
- In one round, a process may send all the messages it requires, while receiving all messages from others
- No message from one round may influence any messages sent within the same round.

Asynchronous System
- There are no bounds on the amount of time a node might take
- There is no global clock nor consistent clock rate
- Each node processes independently of others
- Coordination is achieved via events such as message arrival
First Aspect: Synchrony Mode

Synchronous System
- Assume known bounds on message delays and process speeds
- All communication proceeds in rounds.
- In one round, a process may send all the messages it requires, while receiving all messages from others
- No message from one round may influence any messages sent within the same round.

Asynchronous System
- There are no bounds on the amount of time a node might take
- There is no global clock nor consistent clock rate
- Each node processes independently of others
- Coordination is achieved via events such as message arrival
First Aspect: Synchrony Mode

**Synchronous System**
- Assume known bounds on message delays and process speeds
- All communication proceeds in rounds.
- In one round, a process may send all the messages it requires, while receiving all messages from others.
- No message from one round may influence any messages sent within the same round.

**Asynchronous System**
- There are no bounds on the amount of time a node might take.
- There is no global clock nor consistent clock rate.
- Each node processes independently of others.
- Coordination is achieved via events such as message arrival.

**Partially-Synchronous System**
- Assumes that among the nodes, there is a subset that can communicate in a timely manner.
- Only a limited number of nodes are perceived as arbitrarily slow.
- Reasonable in data centers which are more predictable and controllable than an open Internet environment.
Second Aspect: Failure Model
Second Aspect: Failure Model

- Nodes operate at arbitrary speed
- May fail by stopping, and may restart
- May not collude, lie, or otherwise attempt to subvert the protocol.
Second Aspect: Failure Model

Crash Failure

- Nodes operate at arbitrary speed
- May fail by stopping, and may restart
- May not collude, lie, or otherwise attempt to subvert the protocol.

Byzantine Failure

- Faulty nodes may exhibit arbitrary, potentially malicious, behavior
Second Aspect: Failure Model

Crash Failure
- Nodes operate at arbitrary speed
- May fail by stopping, and may restart
- May not collude, lie, or otherwise attempt to subvert the protocol.

Byzantine Failure
- Faulty nodes may exhibit arbitrary, potentially malicious, behavior

Hybrid Failure
- Some nodes might crash whereas some nodes behave maliciously.
Third Aspect: Processing Strategy
Third Aspect: Processing Strategy

• **Pessimistic**
  - Guarantee from the beginning that all the replicas are identical to each other
  - Robust and designed to tolerate the maximum number of possible concurrent failures

• **Optimistic**
  - Replicas speculatively execute requests without running an agreement protocol to definitively establish the order
  - Replicas can diverge
  - Eventual consistency
Fourth Aspect: Participant Awareness
Fourth Aspect: Participant Awareness

• **Known**
  • The participants are known and identified
  • Assume the maximum number of failures in the system is $f$

• **Unknown**
  • The set of participants is assumed to be unknown
Fifth Aspect: Complexity Metrics
Fifth Aspect: Complexity Metrics

- Number of nodes
- Number of communication phases
- Message complexity
FLP Result

No deterministic 1-crash-robust consensus algorithm exists with asynchronous communication

Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER
Yale University, New Haven, Connecticut

NANCY A. LYNCH
Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON
University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the “Byzantine Generals” problem.

This work was originally presented at the 2nd ACM Symposium on Principles of Database Systems, March 1983.

Authors’ present addresses: M. J. Fischer, Department of Computer Science, Yale University, P.O. Box 2158, Yale Station, New Haven, CT 06520; N. A. Lynch, Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139; M. S. Paterson, Department of Computer Science, University of Warwick, Coventry CV4 7AL, England

FLP Result

- Impossibility of Distributed Consensus with ONE faulty Process
  Asynchronous system; but reliable network
  - Process: Crash failures. Max ONE failure
  - Consensus problem: all non faulty processes agree on the same value \{0, 1\}.

Fault Tolerance

Safety

Liveness
FLP Result

• Impossibility of Distributed Consensus with ONE faulty Process

Asynchronous system; but reliable network
  • Process: Crash failures. Max ONE failure
  • Consensus problem: all non faulty processes agree on the same value \{0, 1\}.
How to circumvent FLP result?

Sacrifice determinism
Randomized Byzantine consensus algorithm

How to circumvent FLP result?

- **Sacrifice determinism**
  - Randomized Byzantine consensus algorithm

- **Adding synchrony assumption**
  - Define bound on message delay, etc.

---

How to circumvent FLP result?

- Sacrifice determinism
  - Randomized Byzantine consensus algorithm

- Adding synchrony assumption
  - Define bound on message delay, etc.

- Adding oracle (failure detector)
  - Adding trusted component

---

How to circumvent FLP result?

- Sacrifice determinism
  - Randomized Byzantine consensus algorithm

- Adding synchrony assumption
  - Define bound on message delay, etc.

- Adding oracle (failure detector)
  - Adding trusted component

- Change the problem domain
  - range of value or set of values

Lower Bounds on Number of Processes
Lower Bounds on Number of Processes

• [Pease, Shostak, Lamport 80] showed 3f+1 lower bound on number of processes for Byzantine Agreement.

• [Dolev 82] showed 2f+1 connectivity bound for BA.

• [Lamport 83] showed 3f+1 lower bound on number of processes for weak BA.

• [Coan, Dolev, Dwork, Stockmeyer 85] showed 3f+1 lower bound for Byzantine firing squad problem.

• [Dolev, Lynch, Pinter, Stark, Weihl 83] claimed 3f+1 bound for approximate BA.

• [Dolev, Halpern, Strong 84] showed 3f+1 lower bound for Byzantine clock synchronization.

• Easy impossibility proofs for distributed consensus problems [Fischer, Lynch, Merritt PODC 85, DC 86]
Equivalent problems to Consensus

Atomic Broadcast
- Hodzilacos and Toueg, 1994
- Chandra and Toueg, 1996
- Cachin et al., 2001

Consensus

State Machine Replication
- Schneider, 1990

Group Membership
- Guerraoui and Schiper, 2001

Non-blocking Atomic Commit
- Guerraoui and Schiper, 2001

Reducible

Related
PAXOS


- Synchronous
- Asynchronous
- Partially-Synchronous
- Crash
- Byzantine
- Hybrid
- Pessimistic
- Optimistic
- Known nodes
- Unknown nodes
- 2f+1 nodes
- 2 phases
- O(N) Complexity

40 Years of Consensus - Amr El-Agawy, D. Abbadi, ICDE2020
Paxos Properties

- Paxos guarantees safety.
  - Consensus is a stable property: once reached it is never violated; the agreed value is not changed.
Paxos Properties

• Paxos guarantees safety.
  • Consensus is a stable property: once reached it is never violated; the agreed value is not changed.

• Paxos does not guarantee liveness.
  • Consensus is reached if “a large enough subnetwork...is non-faulty for a long enough time.”
  • Otherwise Paxos might never terminate.
Paxos Consensus Algorithm

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates
- Upon receiving ‘enough’ acks, leader sends decision asynchronously
Paxos Consensus Algorithm

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates
- Upon receiving ‘enough’ acks, leader sends decision asynchronously
Paxos Consensus Algorithm

• The clients send updates to the leader
• Leader orders the requests and ‘forwards’ to the replicas
• Leader waits to get acknowledgement of the updates
• Upon receiving ‘enough’ acks, leader sends decision asynchronously
**Paxos Consensus Algorithm**

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates
- Upon receiving ‘enough’ acks, leader sends decision asynchronously
Paxos Consensus Algorithm

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates
- Upon receiving ‘enough’ acks, leader sends decision asynchronously
Paxos Consensus Algorithm

• **Leader Election:** Initially, a leader is elected by a quorum of servers
• **Replication:** Leader replicates new updates on quorum of servers
• **Decision:** Propagates decision to all **asynchronously**
Paxos Consensus Algorithm

- **Leader Election:** Initially, a leader is elected by a quorum of servers
- **Replication:** Leader replicates new updates on quorum of servers
- **Decision:** Propagates decision to all *asynchronously*

![Diagram of Paxos Algorithm]

*Update:* The figure shows the flow of information in the Paxos algorithm. Node 1 is the leader and nodes 2, 3, and N are servers. The leader sends updates to the servers, which then propagate the decision asynchronously.
Paxos Consensus Algorithm

- **Leader Election:** Initially, a leader is elected by a quorum of servers
- **Replication:** Leader replicates new updates on quorum of servers
- **Decision:** Propagates decision to all asynchronously
Paxos Consensus Algorithm

- **Leader Election**: Initially, a leader is elected by a quorum of servers.
- **Replication**: Leader replicates new updates on quorum of servers.
- **Decision**: Propagates decision to all asynchronously.
Paxos Consensus Algorithm

- **Leader Election:** Initially, a leader is elected by a quorum of servers
- **Replication:** Leader replicates new updates on quorum of servers
- **Decision:** Propagates decision to all *asynchronously*
Paxos Consensus Algorithm

- **Leader Election:** Initially, a leader is elected by a quorum of servers.
- **Replication:** Leader replicates new updates on quorum of servers.
- **Decision:** Propagates decision to all asynchronously.

Diagram:

```
1  2  3
\  |  /  \
 \ | /   
  N L N
```

- Prepare
- Accept
- Decision

Update flows from 1 to 2, 3, and L. Then, L propagates the decision asynchronously to 2, 3, and N.
Paxos Consensus Algorithm

- **Leader Election**: Initially, a leader is elected by a quorum of servers
- **Replication**: Leader replicates new updates on quorum of servers
- **Decision**: Propagates decision to all asynchronously
Paxos Consensus Algorithm

- **Leader Election**: Initially, a leader is elected by a quorum of servers
- **Replication**: Leader replicates new updates on quorum of servers
- **Decision**: Propagates decision to all asynchronously

---

**Proposer**

**Acceptors**

1. **Prepare**
2. **Accept**
3. **Decision**

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Paxos Consensus Algorithm

- **Leader Election:** Initially, a leader is elected by a quorum of servers.
- **Replication:** Leader replicates new updates on quorum of servers.
- **Decision:** Propagates decision to all asynchronously.
Paxos Consensus Algorithm

- **Leader Election**: Initially, a leader is elected by a quorum of servers.
- **Replication**: Leader replicates new updates on quorum of servers.
- **Decision**: Propagates decision to all *asynchronously*.
Safety Condition

• Any two sets (quorums) of acceptors must have at least one overlapping acceptor

• This way a new leader will know of a value chosen by old leader through the overlapping acceptor
Safety Condition

- Any two sets (quorums) of acceptors must have at least one overlapping acceptor.
- This way a new leader will know of a value chosen by old leader through the overlapping acceptor.
Safety Condition

• Any two sets (quorums) of acceptors must have at least one overlapping acceptor

• This way a new leader will know of a value chosen by old leader through the overlapping acceptor

Majority Quorum

1  2  3  4  5  6  7
Safety Condition

• Any two sets (quorums) of acceptors must have at least one overlapping acceptor
• This way a new leader will know of a value chosen by old leader through the overlapping acceptor
Paxos is Leader-based

• **Ballots** distinguish among values proposed by different leaders
  • Unique, locally monotonically increasing
  • Processes **respond only to leader with highest ballot**
Paxos is Leader-based

- **Ballots** distinguish among values proposed by different leaders
  - Unique, locally monotonically increasing
  - Processes respond only to leader with highest ballot

- Pairs \( \langle \text{num}, \text{process id} \rangle \) that form a total order.

- \( \langle n_1, p_1 \rangle > \langle n_2, p_2 \rangle \)
  - If \( n_1 > n_2 \)
  - Or \( n_1=n_2 \) and \( p_1 > p_2 \)
Paxos is Leader-based

- **Ballots** distinguish among values proposed by different leaders
  - Unique, locally monotonically increasing
  - Processes respond only to leader with highest ballot
- Pairs $\langle \text{num, process id} \rangle$ that form a total order.
- $\langle n_1, p_1 \rangle > \langle n_2, p_2 \rangle$
  - If $n_1 > n_2$
  - Or $n_1 = n_2$ and $p_1 > p_2$
- If latest known ballot is $\langle n, q \rangle$ then
  - $p$ chooses $\langle n+1, p \rangle$
The First Two Phases of Paxos

• Phase 1: **prepare**
  • If you *believe you are the leader*
    • Choose **new unique ballot number**
    • Learn **outcome of all smaller ballots from majority**
The First Two Phases of Paxos

• Phase 1: **prepare**
  • If you *believe you are the leader*
    • Choose new unique ballot number
    • Learn outcome of all smaller ballots from majority

• Phase 2: **accept**
  • Leader *proposes a value* with its ballot number
  • Leader gets majority to *accept* its proposal
  • A value accepted by a majority can be decided
Leader Crash

\[ v \]
Leader Crash

I will ask everyone to join my ballot
Ballot num: <n, 1>
Leader Crash

Majority agreed to be in my ballot! I will propose value \( v \) at ballot \( n,1 \)

Prepare
Leader Crash

Majority agreed to be in my ballot! I will propose value $v$ at ballot $<n,1>$

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Leader Crash

Majority accepted value $v$! Yay! I will update my state machine!

Prepare

Accept
Majority accepted value $v$! Yay!
I will update my state machine!

Leader Crash

Prepare

Accept

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020

89
The value $v$ was chosen! Any new leader must recover $v$!

Need more variables to remember $v$:
- $\text{AcceptVal}$ to indicate the value accepted
- $\text{AcceptNum}$ to indicate the ballot num at which AcceptVal was accepted

Prepare

Accept
Paxos - Variables

BallotNum_i, initially \( \langle 0,0 \rangle \)
- Latest ballot \( p_i \) took part in (phase 1)

AcceptNum_i, initially \( \langle 0,0 \rangle \)
- Latest ballot \( p_i \) accepted a value in (phase 2)

AcceptVal_i, initially \( \perp \)
- Latest accepted value (phase 2)

The original version of these Paxos slides are from Idit Keidar several years ago.
Thank you. Any errors are mine.
Phase I: Prepare - Leader

if leader then

   BallotNum ← (BallotNum.num+1, myId)
   send ("prepare", BallotNum) to all
Phase I: Prepare - Leader

if leader then

BallotNum ← \langle BallotNum.num + 1, myId\rangle
send ("prepare", BallotNum) to all

• Goal: contact other processes, ask them to join this ballot, and get information about possible past decisions
Phase I: Prepare - Cohort

• Upon receive (“prepare”, bal) from $i$
  
  if $bal \geq BallotNum$ then
  
  \hspace{1cm} BallotNum \leftarrow bal
  
  \hspace{1cm} send (“ack”, bal, AcceptNum, AcceptVal) to $i$
Phase I: Prepare - Cohort

- Upon receive ("prepare", bal) from $i$ if $\text{bal} \geq \text{BallotNum}$ then
  BallotNum $\leftarrow$ bal
  send ("ack", bal, AcceptNum, AcceptVal) to $i$

This is a promise not to accept ballots smaller than bal in the future.

This is a higher ballot than my current, I better join it.
Phase I: Prepare - Cohort

• Upon receive ("prepare", bal) from i if bal ≥ BallotNum then
  BallotNum ← bal
  send ("ack", bal, AcceptNum, AcceptVal) to i

  This is a higher ballot than my current, I better join it

  This is a promise not to accept ballots smaller than bal in the future

  Tell the leader about my latest accepted value and what ballot it was accepted in
Phase II: Accept - Leader

Upon receive (“ack”, BallotNum, b, val) from majority

if all vals = ^ then myVal = initial value
else myVal = received val with highest b

send (“accept”, BallotNum, myVal) to all  /* proposal */

The value accepted in the highest ballot might have been decided, I better propose this value
Phase II: Accept - Cohort

Upon receive (\texttt{"accept"}, b, v)

\textbf{if} \ b \geq \text{BallotNum} \textbf{then}

\hspace{1cm} \text{AcceptNum} \leftarrow b; \text{AcceptVal} \leftarrow v \quad /* \text{accept proposal} */

send (\texttt{"accept"}, b, v) to leader (or to all)

Upon receive (\texttt{"accept"}, b, v) from \textit{majority}

\hspace{1cm} decide v

This is not from an old ballot
Liveness

- $S_1$
- $S_2$
- $S_3$
- $S_4$
- $S_5$

(time)
Liveness

S₁  P 3.1
S₂  P 3.1
S₃  P 3.1
S₄  
S₅  

time

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Liveness

| S₁   | P 3.1 |
| S₂   | P 3.1 |
| S₃   | P 3.1 | P 3.5 |
| S₄   | P 3.5 |
| S₅   | P 3.5 |

Time
Liveness

\[
\begin{align*}
S_1 & : P 3.1 & A 3.1 X \\
S_2 & : P 3.1 & A 3.1 X \\
S_3 & : P 3.1 & P 3.5 & A 3.1 X \\
S_4 & : & P 3.5 \\
S_5 & : & P 3.5 \\
\end{align*}
\]
Liveness

| S1 | P 3.1 | A 3.1 X |
| S2 | P 3.1 | A 3.1 X |
| S3 | P 3.1 | P 3.5  |
| S4 | P 3.5 |        |
| S5 | P 3.5 |        |
Liveness

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Liveness

S\_1: P 3.1 \rightarrow A 3.1 \rightarrow X \rightarrow P 4.1

S\_2: P 3.1 \rightarrow A 3.1 \rightarrow X \rightarrow P 4.1

S\_3: P 3.1 \rightarrow P 3.5 \rightarrow X \rightarrow P 4.1 \rightarrow A 3.5 \rightarrow Y

S\_4: P 3.5 \rightarrow A 3.5 \rightarrow Y

S\_5: P 3.5 \rightarrow A 3.5 \rightarrow Y

Time
Liveness

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
## Liveness

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>P 3.1</td>
<td>P 3.1</td>
<td>P 3.1</td>
<td>P 3.5</td>
<td>P 3.5</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>A 3.1 X</td>
<td>A 3.1 X</td>
<td>A 3.1 X</td>
<td>A 3.5 X</td>
<td>A 3.5 X</td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>P 4.1</td>
<td>P 4.1</td>
<td>P 4.1</td>
<td>P 5.5</td>
<td>P 5.5</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A 4.1 X</td>
<td>A 4.1 X</td>
<td>A 4.1 X</td>
<td>A 4.1 X</td>
<td>A 4.1 X</td>
<td></td>
</tr>
</tbody>
</table>

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Liveness

\[ S_1 \quad P \ 3.1 \quad A \ 3.1 \ X \quad P \ 4.1 \quad A \ 4.1 \ X \]

\[ S_2 \quad P \ 3.1 \quad A \ 3.1 \ X \quad P \ 4.1 \quad A \ 4.1 \ X \]

\[ S_3 \quad P \ 3.1 \quad P \ 3.5 \quad A \ 3.1 \ X \quad P \ 4.1 \quad A \ 3.5 \ X \quad P \ 5.5 \quad A \ 4.1 \ X \]

\[ S_4 \quad P \ 3.5 \quad A \ 3.5 \ Y \quad P \ 5.5 \]

\[ S_5 \quad P \ 3.5 \quad A \ 3.5 \ Y \quad P \ 5.5 \]

time
Liveness

• Competing proposers can **livelock**:

\[
\begin{align*}
S_1 & : P 3.1 & A 3.1 X & P 4.1 & A 4.1 X \\
S_2 & : P 3.1 & A 3.1 X & P 4.1 & A 4.1 X \\
S_3 & : P 3.1 & P 3.5 & A 3.1 X & P 4.1 & A 3.5 X & P 5.5 & A 4.1 X \\
S_4 & : P 3.5 & A 3.5 Y & P 5.5 & A 3.5 Y & P 5.5 \\
S_5 & : P 3.5 & A 3.5 Y & P 5.5 & A 3.5 Y & P 5.5
\end{align*}
\]

• **One solution**: randomized delay before restarting
  • Give other proposers a chance to finish choosing
Recall: Paxos Consensus Algorithm

- **Leader Election:** Initially, a leader is elected by a quorum of servers.
- **Replication:** Leader replicates new updates on quorum of servers.
- **Decision:** Propagates decision to all asynchronously.
Recall: Paxos Consensus Algorithm

- **Leader Election**: Initially, a leader is elected by a quorum of servers.
- **Replication**: Leader replicates new updates on quorum of servers.
- **Decision**: Propagates decision to all *asynchronously*.

![Paxos Consensus Algorithm Diagram]

Why is this phase needed?
Observation

• In **Phase 1**, no consensus values are sent:
  • Leader chooses largest unique ballot number
  • Gets a majority to “vote” for this ballot number
  • Learns the outcome of all smaller ballots

• In **Phase 2**, leader proposes its own initial value or latest value it learned in Phase 1
Optimization

• Run Phase 1 only when the leader changes
  • Phase 1 is called “view change” or “recovery mode”
  • Phase 2 is the “normal mode”

• Each message includes BallotNum (from the last Phase 1) and ReqNum
• Respond only to messages with the “right” BallotNum
Recall: Paxos Consensus Algorithm

- **Leader Election**: Initially, a leader is elected by a quorum of servers.
- **Replication**: Leader replicates new updates on quorum of servers.
- **Decision**: Propagates decision to all asynchronously.

![Paxos Consensus Algorithm Diagram]

Leader Election

1. Prepare
2. Accept
3. Decision
Recall: Paxos Consensus Algorithm

- **Leader Election:** Initially, a leader is elected by a quorum of servers
- **Replication:** Leader replicates new updates on quorum of servers
- **Decision:** Propagates decision to all asynchronously
Multi-Paxos

- Separate instance of Basic Paxos for each log entry:
  - Add index argument to Prepare and Accept (selects entry in log)
Multi-Paxos

• Separate instance of Basic Paxos for each log entry:
  • Add **index** argument to Prepare and Accept (selects entry in log)

1. Client sends command to server
Multi-Paxos

• Separate instance of Basic Paxos for each log entry:
  • Add index argument to Prepare and Accept (selects entry in log)

1. Client sends command to server

2. Server uses Paxos to choose command as value for a log entry
Multi-Paxos

- Separate instance of Basic Paxos for each log entry:
  - Add `index` argument to Prepare and Accept (selects entry in log)

1. Client sends command to server

2. Server uses Paxos to choose command as value for a log entry

3. Server waits for previous log entries to be applied, then applies new command to state machine
Multi-Paxos

• Separate instance of Basic Paxos for each log entry:
  • Add index argument to Prepare and Accept (selects entry in log)

1. Client sends command to server
2. Server uses Paxos to choose command as value for a log entry
3. Server waits for previous log entries to be applied, then applies new command to state machine
4. Server returns result from state machine to client
Raft

- Equivalent to Paxos in fault-tolerance
- Meant to be more understandable
- Uses a leader approach
- Integrates consensus with log management

Ongaro, D., & Ousterhout, J. In search of an understandable consensus algorithm. In USENIX ATC, 2014

<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Crash</th>
<th>2f+1 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous</td>
<td>Byzantine</td>
<td>2 phases</td>
</tr>
<tr>
<td>Partially-Synchronous</td>
<td>Pessimistic</td>
<td>Unknown nodes</td>
</tr>
<tr>
<td></td>
<td>Optimistic</td>
<td>O(N) Complexity</td>
</tr>
<tr>
<td></td>
<td>Known nodes</td>
<td></td>
</tr>
</tbody>
</table>

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Abstract Paxos
Abstract Paxos
Abstract Paxos
Two Phase Commit

Two Phase Commit

- A distributed transaction accesses data stored across multiple servers
- 2PC is *atomic commitment* protocol: either all servers commit or no server commits
Two Phase Commit

- A distributed transaction accesses data stored across multiple servers
- 2PC is *atomic commitment* protocol: either all servers commit or no server commits
Two Phase Commit

• A distributed transaction accesses data stored across multiple servers
• 2PC is *atomic commitment* protocol: either all servers commit or no server commits
Two Phase Commit

- A distributed transaction accesses data stored across multiple servers
- 2PC is *atomic commitment* protocol: either all servers commit or no server commits
Two Phase Commit

- A distributed transaction accesses data stored across multiple servers
- 2PC is *atomic commitment* protocol: either all servers commit or no server commits
Abstract 2PC
Abstract 2PC

Value Discovery → Make Fault-tolerant

1. 2. 3. 4. 5. 6.

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Abstract 2PC
Abstract 3PC

• 2PC has possibility of **Blocking**
• 3 Phase Commit: Replicate decision to cohorts (like Paxos)
Abstract 3PC

• 2PC has possibility of **Blocking**
• 3 Phase Commit: **Replicate decision to cohorts (like Paxos)**
Abstract 3PC

• 2PC has possibility of **Blocking**
• 3 Phase Commit: **Replicate decision to cohorts (like Paxos)**
Abstract 3PC

• 2PC has possibility of **Blocking**
• 3 Phase Commit: Replicate decision to cohorts (like Paxos)
Fault-tolerant 3PC (with Termination)

- If leader fails: Elect new leader and execute termination protocol
Fault-tolerant 3PC (with Termination)

• If leader fails: Elect new leader and execute termination protocol
Fault-tolerant 3PC (with Termination)

• If leader fails: Elect new leader and execute termination protocol
Fault-tolerant 3PC (with Termination)

- If leader fails: **Elect new leader** and execute termination protocol
Common phases observed?

- Paxos and 2PC/3PC are leader-based protocols
Common phases observed?

- Paxos and 2PC/3PC are leader-based protocols
- Agreement on a single value is the main goal
Common phases observed?

- Paxos and 2PC/3PC are leader-based protocols
- Agreement on a single value is the main goal
- Both protocols ensure fault tolerance on the decided value
Paxos and 2PC/3PC are leader-based protocols.

Agreement on a single value is the main goal.

Both protocols ensure fault tolerance on the decided value.

Disseminate the decision, typically asynchronously.
Leader Election

Consensus & Commitment (C&C) Framework

Consensus & Commitment (C&C) Framework

Consensus & Commitment (C&C) Framework

Leader Election → Value Discovery → Fault-tolerant Agreement → Decision

C&C Framework

• A pedagogical tool to understand many existing protocols
• Helps us develop insights for protocols in novel settings
C&C Framework

• A pedagogical tool to understand many existing protocols
• Helps us develop insights for protocols in novel settings

Three Phase Commit
C&C Framework

• A pedagogical tool to understand many existing protocols
• Helps us develop insights for protocols in novel settings

Three Phase Commit

Leader Election

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
C&C Framework

• A pedagogical tool to understand many existing protocols
• Helps us develop insights for protocols in novel settings

Three Phase Commit

Leader Election -> Value Discovery

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
C&C Framework

- A pedagogical tool to understand many existing protocols
- Helps us develop insights for protocols in novel settings

Three Phase Commit

1. Leader Election
2. Value Discovery
3. Fault-tolerant Agreement
C&C Framework

• A pedagogical tool to understand many existing protocols
• Helps us develop insights for protocols in novel settings

Three Phase Commit

Leader Election → Value Discovery → Fault-tolerant Agreement → Decision

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
C&C Framework

- A pedagogical tool to understand many existing protocols
- Helps us develop insights for protocols in novel settings

Three Phase Commit

Leader Election → Value Discovery → Fault-tolerant Agreement → Decision

Paxos

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
C&C Framework

- A pedagogical tool to understand many existing protocols
- Helps us develop insights for protocols in novel settings

Three Phase Commit

- Leader Election
- Value Discovery
- Fault-tolerant Agreement
- Decision

Paxos

- Leader Election

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
C&C Framework

- A pedagogical tool to understand many existing protocols
- Helps us develop insights for protocols in novel settings

Three Phase Commit

1. Leader Election
2. Value Discovery
3. Fault-tolerant Agreement
4. Decision

Paxos

1. Leader Election
2. Value Discovery

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
C&C Framework

- A pedagogical tool to understand many existing protocols
- Helps us develop insights for protocols in novel settings

Three Phase Commit

1. Leader Election
2. Value Discovery
3. Fault-tolerant Agreement
4. Decision

Paxos

1. Leader Election
2. Value Discovery
3. Fault-tolerant Agreement

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
C&C Framework

- A pedagogical tool to understand many existing protocols
- Helps us develop insights for protocols in novel settings

Three Phase Commit

1. Leader Election
2. Value Discovery
3. Fault-tolerant Agreement
4. Decision

Paxos

1. Leader Election
2. Value Discovery
3. Fault-tolerant Agreement
4. Decision

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Fast Paxos

Reduce messages delays
Sacrifice quorum size

Synchronous
Asynchronous
Partially-Synchronous
Crash
Byzantine
Hybrid
Pessimistic
Optimistic
Known nodes
Unknown nodes
3f+1 nodes
1 or 3 phases
O(N) Complexity

Fast Paxos

- Generalizes Basic Paxos to reduce end-to-end message delays.
- Basic Paxos: 3 message delays from client request to learning
- Fast Paxos allows 2 message delays where
  1. the system includes $3f+1$ nodes (instead of $2f+1$)
  2. the Client sends its request to multiple destinations.

- Intuition:
  - If the leader has no value to propose, a client sends an Accept! to all nodes.
  - Backups respond as in Basic Paxos, sending Accepted messages to the leader
Fast Round

<table>
<thead>
<tr>
<th>Client</th>
<th>AnyMsg</th>
<th>Accept!</th>
<th>Accepted</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>replica 0 (Leader)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fast Round

Any Message enables a backup to select its own value (proposed by a client)
Fast Round

Any Message enables a backup to select its own value (proposed by a client)
Fast Round

Any Message enables a backup to select its own value (proposed by a client)
Fast Round

Any Message enables a backup to select its own value (proposed by a client)

AnyMsg Accept! Accepted Commit

Client replica 0 (Leader) replica 1 replica 2 replica 3

only one value is accepted
Any Message enables a backup to select its own value (proposed by a client)
Collision Happens!

<table>
<thead>
<tr>
<th>AnyMsg</th>
<th>Accept!</th>
<th>Accepted</th>
<th>Accept!</th>
<th>Accepted</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client 1</td>
<td>AnyMsg</td>
<td>Accept!</td>
<td>Accepted</td>
<td>Accept!</td>
<td>Accepted</td>
</tr>
<tr>
<td>Client 2</td>
<td>AnyMsg</td>
<td>Accept!</td>
<td>Accepted</td>
<td>Accept!</td>
<td>Accepted</td>
</tr>
</tbody>
</table>

replica 0 (Leader)
replica 1
replica 2
replica 3

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Collision Happens!

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Collision Happens!

AnyMsg  Accept!  Accepted  Accept!  Accepted  Commit

Client 1
Client 2

replica 0
(Leader)
replica 1
replica 2
replica 3

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Collision Happens!

Choose the value with the majority quorum if exists.
Collision Happens!

Chooses the value with the majority quorum if exists
Collision Happens!

Chooses the value with the majority quorum if exists
Collision Happens!

AnyMsg Accept! Accepted Accept! Accepted Commit

Client 1
Client 2

replica 0
(Leader)
replica 1
replica 2
replica 3

different values are accepted

Chooses the value with the majority quorum if exists

Classic Round

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Flexible Paxos

Howard, H., Malkhi, D., & Spiegelman, A. Flexible Paxos: Quorum Intersection Revisited. In *OPODIS, 2017*

It is not necessary to require all quorums in Paxos to intersect

<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Asynchronous</th>
<th>Partially-Synchronous</th>
<th>Crash</th>
<th>Byzantine</th>
<th>Hybrid</th>
<th>Pessimistic</th>
<th>Optimistic</th>
<th>Known nodes</th>
<th>Unknown nodes</th>
<th>2f+1 nodes</th>
<th>2 phases</th>
<th>O(N) Complexity</th>
</tr>
</thead>
</table>

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Flexible Paxos

• Majority quorums for BOTH Leader Election AND Replication are too conservative

• Generalized Quorum Condition: only Leader Election Quorums and Replication Quorums must intersect.
  • Decouple Leader Election Quorums from Replication Quorums
  • Arbitrarily small replication quorums as long as Leader Election Quorums intersect with every Replication Quorum

• No changes to Paxos algorithms
Flexible Paxos

• Majority quorums for BOTH Leader Election AND Replication are too conservative

• Generalized Quorum Condition: only Leader Election Quorums and Replication Quorums must intersect.
  • Decouple Leader Election Quorums from Replication Quorums
  • Arbitrarily small replication quorums as long as Leader Election Quorums intersect with every Replication Quorum

• No changes to Paxos algorithms
Flexible Paxos

• Majority quorums for BOTH Leader Election AND Replication are too conservative

• Generalized Quorum Condition: only Leader Election Quorums and Replication Quorums must intersect.
  • Decouple Leader Election Quorums from Replication Quorums
  • Arbitrarily small replication quorums as long as Leader Election Quorums intersect with every Replication Quorum

• No changes to Paxos algorithms
Flexible Paxos

• Majority quorums for BOTH Leader Election AND Replication are too conservative

• Generalized Quorum Condition: only Leader Election Quorums and Replication Quorums must intersect.
  • Decouple Leader Election Quorums from Replication Quorums
  • Arbitrarily small replication quorums as long as Leader Election Quorums intersect with every Replication Quorum

• No changes to Paxos algorithms
Paxos in Real Systems

Google MegaStore, Clustrix, Ceph, Google Cloud Spanner, Bing, OpenReplica, Doozerd, Chubby, wandisco, neo4j, XtreemFS
Google Spanner
Google Spanner

Application Access Tier

Datacenter A  Datacenter B  Datacenter Z

Storage Tier
Abstract Replication
PAXOS

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Google Spanner

Application Access Tier

Application Execution Tier

Transactions

2PL+2PC

Datacenter A

Datacenter B

Datacenter Z

Storage Tier

Abstract Replication

PAXOS

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Google Bigtable

Master

Control Operations

Lease Management

Master and Chubby Proxies

T₁, T₂, ..., Tₙ

Tablets

Tablet Server

Cache Manager

Log Manager

Tablet Server

Google File System
Google Bigtable

• A persistent and distributed lock service
• Consists of 5 replicas
• Uses Paxos to keep copies consistent
What if nodes behave maliciously?!
Reaching Agreement in the Presence of Fault

Reaching Agreement in the Presence of Fault
Reaching Agreement in the Presence of Fault

• In a system with $f$ faulty processes, an agreement can be achieved only if $2f+1$ correctly functioning processes are present, for a total of $3f+1$.

• i.e., An agreement is possible only if more than two-thirds of the processes are working properly.
Reaching Agreement in the Presence of Fault

• In a system with \( f \) faulty processes, an agreement can be achieved only if \( 2f+1 \) correctly functioning processes are present, for a total of \( 3f+1 \).

• i.e., An agreement is possible only if more than two-thirds of the processes are working properly.

• Model:
  • Processes are synchronous
  • Messages are unicast while preserving ordering
  • Communication delay is bounded
  • There are \( N \) processes, where each process \( i \) will provide a value \( v_i \) to the others
  • There are at most \( f \) faulty processes
Reaching Agreement in the Presence of Fault

• In a system with \( f \) faulty processes, an agreement can be achieved only if \( 2f+1 \) correctly functioning processes are present, for a total of \( 3f+1 \).

• i.e., An agreement is possible only if more than two-thirds of the processes are working properly.

• Model:
  • Processes are synchronous
  • Messages are unicast while preserving ordering
  • Communication delay is bounded
  • There are \( N \) processes, where each process \( i \) will provide a value \( v_i \) to the others
  • There are at most \( f \) faulty processes

• Each process \( i \) constructs a vector \( V \) of length \( N \), such that
  • If process \( i \) is non-faulty, \( V[i] = i \)
  • Otherwise, \( V[i] \) is undefined
Reaching Agreement in the Presence of Fault

• Case I: $N = 4$ and $f = 1$
Reaching Agreement in the Presence of Fault

• Case I: \( N = 4 \) and \( f = 1 \)

**Step1:** Each process sends its value to the others

1  2  3  4

Faulty process
Reaching Agreement in the Presence of Fault

- Case I: $N = 4$ and $f = 1$

**Step 1:** Each process sends its value to the others
Reaching Agreement in the Presence of Fault

• Case I: $N = 4$ and $f = 1$

**Step 1:** Each process sends its value to the others
Reaching Agreement in the Presence of Fault

• Case I: $N = 4$ and $f = 1$

**Step 1:** Each process sends its value to the others
Reaching Agreement in the Presence of Fault

• Case I: $N = 4$ and $f = 1$

Step 1: Each process sends its value to the others

Faulty process
Reaching Agreement in the Presence of Fault

• Case I: $N = 4$ and $f = 1$

**Step 1:** Each process sends its value to the others

Faulty process
Reaching Agreement in the Presence of Fault

• Case I: \( N = 4 \) and \( f = 1 \)

**Step 1:** Each process sends its value to the others

1. \( \text{Got}(1, 2, x, 4) \)
2. \( \text{Got}(1, 2, y, 4) \)
3. \( \text{Got}(1, 2, 3, 4) \)
4. \( \text{Got}(1, 2, z, 4) \)

**Step 2:** Each process collects values received in a vector

Faulty process
Reaching Agreement in the Presence of Fault

- Case I: $N = 4$ and $f = 1$

**Step 1:** Each process sends its value to the others

1. Process 1 sends value 1 to processes 2, 3, and 4.
2. Process 2 sends value 2 to processes 1, 3, and 4.
3. Process 3 sends value 3 to processes 1, 2, and 4.
4. Process 4 sends value 4 to processes 1, 2, and 3.

**Step 2:** Each process collects values received in a vector

- Process 1: Got(1, 2, x, 4)
- Process 2: Got(1, 2, y, 4)
- Process 3: Got(1, 2, 3, 4)
- Process 4: Got(1, 2, z, 4)
Reaching Agreement in the Presence of Fault

• Case I: \( N = 4 \) and \( f = 1 \)

**Step1:** Each process sends its value to the others

**Step2:** Each process collects values received in a vector

**Step3:** Every process passes its vector to every other process

```
Faulty process
```

```
1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)
```
Reaching Agreement in the Presence of Fault

• Case I: \( N = 4 \) and \( f = 1 \)

**Step 1:** Each process sends its value to the others

**Step 2:** Each process collects values received in a vector

**Step 3:** Every process passes its vector to every other process

Faulty process

1. Got(1, 2, x, 4)
2. Got(1, 2, y, 4)
3. Got(1, 2, 3, 4)
4. Got(1, 2, z, 4)

1. Got(1, 2, y, 4)
2. Got(1, 2, z, 4)
3. Got(1, 2, y, 4)
4. Got(1, 2, z, 4)
Reaching Agreement in the Presence of Fault

• Case I: \( N = 4 \) and \( f = 1 \)

**Step 1:** Each process sends its value to the others

- Got(1, 2, \( x \), 4)
- Got(1, 2, \( y \), 4)
- Got(1, 2, 3, 4)
- Got(1, 2, \( z \), 4)

**Step 2:** Each process collects values received in a vector

- 1 Got(1, 2, \( x \), 4)
- 2 Got(1, 2, \( y \), 4)
- 3 Got(1, 2, 3, 4)
- 4 Got(1, 2, \( z \), 4)

**Step 3:** Every process passes its vector to every other process

- 1 Got (1, 2, \( y \), 4)
- 2 Got (1, 2, \( x \), 4)
- (a, b, c, d)
- (1, 2, \( z \), 4)
- (1, 2, \( z \), 4)
Reaching Agreement in the Presence of Fault

• Case I: $N = 4$ and $f = 1$

**Step 1:** Each process sends its value to the others

1. Step 1: Each process sends its value to the others
   - Got(1, 2, x, 4)
   - Got(1, 2, y, 4)
   - Got(1, 2, 3, 4)
   - Got(1, 2, z, 4)

**Step 2:** Each process collects values received in a vector

1. Got(1, 2, y, 4) (a, b, c, d)
2. Got(1, 2, z, 4) (e, f, g, h)
3. Got(1, 2, x, 4) (i, j, k, l)

**Step 3:** Every process passes its vector to every other process

1. Got(1, 2, y, 4) (a, b, c, d)
2. Got(1, 2, z, 4) (e, f, g, h)
3. Got(1, 2, x, 4) (i, j, k, l)
Reaching Agreement in the Presence of Fault

Step 4:
• Each process examines the $i$-th element of each of the newly received vectors.
• If any value has a majority, that value is put into the result vector.
• If no value has a majority, the corresponding element of the result vector is marked **UNKNOWN**.
Reaching Agreement in the Presence of Fault

**Step 4:**
- Each process examines the $i$-th element of each of the newly received vectors.
- If any value has a *majority*, that value is put into the result vector.
- If no value has a majority, the corresponding element of the result vector is marked **UNKNOWN**.

1. Got

   (1, 2, y, 4)
   (a, b, c, d)
   (1, 2, z, 4)
Reaching Agreement in the Presence of Fault

Step 4:
- Each process examines the $i$-th element of each of the newly received vectors
- If any value has a majority, that value is put into the result vector
- If no value has a majority, the corresponding element of the result vector is marked **UNKNOWN**

1 Got

- $(1, 2, y, 4)$
- $(a, b, c, d)$
- $(1, 2, z, 4)$

**Result Vector:**
- $(1, 2, \text{UNKNOWN}, 4)$
Reaching Agreement in the Presence of Fault

**Step 4:**
- Each process examines the \( i \)-th element of each of the newly received vectors.
- If any value has a majority, that value is put into the result vector.
- If no value has a majority, the corresponding element of the result vector is marked **UNKNOWN**.

<table>
<thead>
<tr>
<th>Got 1</th>
<th>Got 2</th>
</tr>
</thead>
</table>
| (1, 2, y, 4)  
(a, b, c, d)  
(1, 2, z, 4) | (1, 2, x, 4)  
(e, f, g, h)  
(1, 2, z, 4) |

**Result Vector:**
(1, 2, **UNKNOWN**, 4)
Reaching Agreement in the Presence of Fault

**Step 4:**
- Each process examines the $i$-th element of each of the newly received vectors.
- If any value has a majority, that value is put into the result vector.
- If no value has a majority, the corresponding element of the result vector is marked **UNKNOWN**.

1. Got
   - (1, 2, y, 4)
   - (a, b, c, d)
   - (1, 2, z, 4)

   **Result Vector:**
   - (1, 2, **UNKNOWN**, 4)

2. Got
   - (1, 2, x, 4)
   - (e, f, g, h)
   - (1, 2, z, 4)

   **Result Vector:**
   - (1, 2, **UNKNOWN**, 4)
## Reaching Agreement in the Presence of Fault

### Step 4:
- Each process examines the $i$-th element of each of the newly received vectors.
- If any value has a majority, that value is put into the result vector.
- If no value has a majority, the corresponding element of the result vector is marked `UNKNOWN`.

<table>
<thead>
<tr>
<th>Step</th>
<th>Got</th>
<th>Result Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1, 2, y, 4) (a, b, c, d) (1, 2, z, 4)</td>
<td>(1, 2, UNKNOWN, 4)</td>
</tr>
<tr>
<td>2</td>
<td>(1, 2, x, 4) (e, f, g, h) (1, 2, z, 4)</td>
<td>(1, 2, UNKNOWN, 4)</td>
</tr>
<tr>
<td>4</td>
<td>(1, 2, x, 4) (1, 2, y, 4) (i, j, k, l)</td>
<td></td>
</tr>
</tbody>
</table>
Reaching Agreement in the Presence of Fault

**Step 4:**
- Each process examines the $i$-th element of each of the newly received vectors.
- If any value has a majority, that value is put into the result vector.
- If no value has a majority, the corresponding element of the result vector is marked **UNKNOWN**.

1 Got

(1, 2, y, 4)
(a, b, c, d)
(1, 2, z, 4)

Result Vector: (1, 2, **UNKNOWN**, 4)

2 Got

(1, 2, x, 4)
(e, f, g, h)
(1, 2, z, 4)

Result Vector: (1, 2, **UNKNOWN**, 4)

4 Got

(1, 2, x, 4)
(1, 2, y, 4)
(i, j, k, l)

Result Vector: (1, 2, **UNKNOWN**, 4)
Reaching Agreement in the Presence of Fault

**Step 4:**
- Each process examines the *i*-th element of each of the newly received vectors.
- If any value has a **majority**, that value is put into the result vector.
- If no value has a majority, the corresponding element of the result vector is marked **UNKNOWN**.

<table>
<thead>
<tr>
<th>Got</th>
<th>Result Vector:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1, 2, y, 4)</td>
</tr>
<tr>
<td></td>
<td>(a, b, c, d)</td>
</tr>
<tr>
<td></td>
<td>(1, 2, z, 4)</td>
</tr>
<tr>
<td>2</td>
<td>(1, 2, x, 4)</td>
</tr>
<tr>
<td></td>
<td>(e, f, g, h)</td>
</tr>
<tr>
<td></td>
<td>(1, 2, z, 4)</td>
</tr>
<tr>
<td>4</td>
<td>(1, 2, x, 4)</td>
</tr>
<tr>
<td></td>
<td>(1, 2, y, 4)</td>
</tr>
<tr>
<td></td>
<td>(i, j, k, l)</td>
</tr>
</tbody>
</table>

**Result Vector:**
(1, 2, UNKNOWN, 4)

**Is 3f+1 optimal?**

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Reaching Agreement in the Presence of Fault

• Case II: $N = 3$ and $f = 1$

**Step 1:** Each process sends its value to the others

1. Got(1, 2, x)
2. Got(1, 2, y)
3. Got(1, 2, 3)

**Step 2:** Each process collects values received in a vector

1. Got(1, 2, y) (a, b, c)
2. Got(1, 2, x) (d, e, f)

**Step 3:** Every process passes its vector to every other process

Faulty process
Reaching Agreement in the Presence of Fault

**Step 4:**
- Each process examines the $i$th element of each of the newly received vectors.
- If any value has a *majority*, that value is put into the result vector.
- If no value has a majority, the corresponding element of the result vector is marked **UNKNOWN**.

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Reaching Agreement in the Presence of Fault

Step 4:
- Each process examines the $i$th element of each of the newly received vectors
- If any value has a majority, that value is put into the result vector
- If no value has a majority, the corresponding element of the result vector is marked UNKNOWN

1 Got
(1, 2, y)
(a, b, c)
Reaching Agreement in the Presence of Fault

**Step 4:**
- Each process examines the $i$th element of each of the newly received vectors.
- If any value has a *majority*, that value is put into the result vector.
- If no value has a majority, the corresponding element of the result vector is marked **UNKNOWN**.

---

1. Got

   - (1, 2, y)
   - (a, b, c)

Result Vector:

- (UNKNOWN, UNKNOWN, UNKNOWN)
Reaching Agreement in the Presence of Fault

Step 4:
- Each process examines the $i$th element of each of the newly received vectors
- If any value has a majority, that value is put into the result vector
- If no value has a majority, the corresponding element of the result vector is marked UNKNOWN

Result Vector:
(UNKNOWN, UNKNOWN, UNKNOWN)
Reaching Agreement in the Presence of Fault

**Step 4:**
- Each process examines the \( i \)th element of each of the newly received vectors.
- If any value has a **majority**, that value is put into the result vector.
- If no value has a majority, the corresponding element of the result vector is marked **UNKNOWN**.

1 Got

\[(1, 2, y)\]
\[(a, b, c)\]

2 Got

\[(1, 2, x)\]
\[(d, e, f)\]

Result Vector:

\[(\text{UNKNOWN}, \text{UNKNOWN}, \text{UNKNOWN})\]

Result Vector:

\[(\text{UNKNOWN}, \text{UNKNOWN}, \text{UNKNOWN})\]
Reaching Agreement in the Presence of Fault

Step 4:
• Each process examines the $ith$ element of each of the newly received vectors
• If any value has a majority, that value is put into the result vector
• If no value has a majority, the corresponding element of the result vector is marked UNKNOWN

1 Got
(1, 2, y)
(a, b, c)

2 Got
(1, 2, x)
(d, e, f)

Result Vector:
(UNKNOWN, UNKNOWN, UNKNOWN)
Practical Byzantine Fault Tolerance

Castro, Miguel, and Barbara Liskov.
"Practical Byzantine fault tolerance."
- OSDI, vol. 99, 1999
- ACM Transactions on Computer Systems, 2002

<table>
<thead>
<tr>
<th>Synchronous</th>
<th>Crash</th>
<th>3f+1 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous</td>
<td>Byzantine</td>
<td>3 phases</td>
</tr>
<tr>
<td>Partially-Synchronous</td>
<td>Hybrid</td>
<td>Unknown nodes</td>
</tr>
</tbody>
</table>

Pessimistic  Known nodes  O(N²) Complexity
Optimistic   Unknown nodes
Why doesn’t Paxos work with Byzantine nodes?

• Cannot rely on the primary to assign sequence number
  • A Malicious primary can assign the same sequence to different requests!

• Cannot use Paxos for leader election
  • Paxos uses a majority \((f+1)\) accept-quorum to tolerate \(f\) benign faults out of \(2f+1\) nodes
  • Does the intersection of two quorums always contain one honest node?
  • Bad node tells different things to different quorums!
    • E.g. tell N1 accept=\(\text{val1}\) and tell N2 accept=\(\text{val2}\)
PBFT Main Ideas

• Configuration
  • Use 3f+1 nodes

• To deal with malicious primary
  • Use a 3-phase protocol

• To deal with loss of agreement
  • Use a bigger quorum (2f+1 out of 3f+1 nodes)

• Need to authenticate communications
Failure Assumption
Failure Assumption

• N > 3f    Why?

• To make any progress must be able to tolerate f failures, i.e., must be able to make progress if only n-f processes respond.

• BUT maybe the f that did not respond are not faulty, but slow (asynchronous systems), and among n-f that responded f are faulty!

• Must have enough responses from non-faulty to outnumber faulty
  • n-2f > f  →  n>3f
Failure Assumption

• \( N > 3f \)  Why?

• To make any progress must be able to tolerate \( f \) failures, i.e., must be able to make progress if only \( n-f \) processes respond.

• BUT maybe the \( f \) that did not respond are not faulty, but slow (asynchronous systems), and among \( n-f \) that responded \( f \) are faulty!

• Must have enough responses from non-faulty to outnumber faulty
  • \( n-2f > f \)  \( \rightarrow \)  \( n > 3f \)
Quorum and Network Size

• $N > 3f$  Why? (Another Argument!)

• Any two Quorums of responses $Q$ need to intersect in at least $f+1$ nodes
  • $Q_1 + Q_2 > N + f$
  • $(N-f) + (N-f) > N + f  \Rightarrow N > 3f$
Quorum and Network Size

• $N > 3f$  Why? (Another Argument!)

• Any two Quorums of responses $Q$ need to intersect in at least $f+1$ nodes
  • $Q_1 + Q_2 > N + f$
  • $(N-f) + (N-f) > N + f \implies N > 3f$

3f+1 replicas
quorums have at least 2f+1 replicas
quorums intersect in at least one correct replica
(Multi-)Paxos Review

Let’s assume the leader is already elected
(Multi-)Paxos Review

• Let’s assume the leader is already elected
(Multi-)Paxos Review

• Let’s assume the leader is already elected
(Multi-)Paxos Review

• Let’s assume the leader is already elected
(Multi-)Paxos Review

• Let’s assume the leader is already elected

If the primary receives $f$ Matching accepted messages (including itself $f+1$), it sends commit and reply.
(Multi-)Paxos with Malicious Backups

<table>
<thead>
<tr>
<th>Request</th>
<th>Accept</th>
<th>Accepted</th>
<th>Commit+ Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>replica 0 (Primary)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 3</td>
<td>fail</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(Multi-)Paxos with Malicious Backups

• What if $f$ of the backups (not the primary) are malicious?!  
  • $3f+1$ nodes needed!

<table>
<thead>
<tr>
<th>Request</th>
<th>Accept</th>
<th>Accepted</th>
<th>Commit+ Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>replica 0 (Primary)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 3</td>
<td></td>
<td></td>
<td>fail</td>
</tr>
</tbody>
</table>
(Multi-)Paxos with Malicious Backups

• What if $f$ of the backups (not the primary) are malicious?!
  • $3f+1$ nodes needed!

```
<table>
<thead>
<tr>
<th>Request</th>
<th>Accept</th>
<th>Accepted</th>
<th>Commit+ Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>replica 0 (Primary)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 3</td>
<td></td>
<td>fail</td>
<td></td>
</tr>
</tbody>
</table>
```
(Multi-)Paxos with Malicious Backups

• What if $f$ of the backups (not the primary) are malicious?!  
  • $3f+1$ nodes needed!

```
replica 0  (Primary)
replica 1
replica 2
replica 3  fail
```
(Multi-)Paxos with Malicious Backups

• What if $f$ of the backups (not the primary) are malicious?!  
  • $3f+1$ nodes needed!

Request  | Accept  | Accepted  | Commit+Reply
---|---|---|---
replica 0 (Primary) | | | 
replica 1 | | | 
replica 2 | | | 
replica 3 | | | fail
(Multi-)Paxos with Malicious Backups

• What if $f$ of the backups (not the primary) are malicious?!
  • $3f+1$ nodes needed!

```
Request  Accept  Accepted  Commit+ Reply
replica 0 (Primary)
replica 1
replica 2
replica 3 fail

the primary waits for $2f$
Matching accepted messages (including itself $2f+1$)
```
(Multi-)Paxos Optimization

• Can nodes commit earlier?!
(Multi-)Paxos Optimization

• Can nodes commit earlier?!
(Multi-)Paxos Optimization

• Can nodes commit earlier?!
(Multi-)Paxos Optimization

• Can nodes commit earlier?!

If a replica receives $2f+1$ matching commit messages, it sends a reply.
(Multi-)Paxos Optimization

- Can nodes commit earlier?!

Client waits for f+1 matching replies.

If a replica receives 2f+1 matching commit messages, it sends a reply.

Request | Accept | Commit | Reply

replica 0 (Primary)

replica 1

replica 2

replica 3 fail
From (Multi-)Paxos to PBFT
From (Multi-)Paxos to PBFT

• When a replica receives an *Accept* message, it knows every replica receives the same message

• What if the leader is malicious?!!
  • Assigns different sequence numbers to the same request
  • Assigns the same sequence number to different request

• One more phase of communication is needed!
From (Multi-)Paxos to PBFT

• When a replica receives an Accept message, it knows every replica receives the same message

• What if the leader is malicious?!!
  • Assigns different sequence numbers to the same request
  • Assigns the same sequence number to different request

• One more phase of communication is needed!
Normal Case Operation
Normal Case Operation

• The algorithm has three phases:
  • *pre-prepare* picks order of requests
  • *prepare* ensures order within views
  • *commit* ensures order across views
Normal Case Operation

• The algorithm has three phases:
  • *pre-prepare* picks order of requests
  • *prepare* ensures order within views
  • *commit* ensures order across views

• A replica executes a request m if
  • m is *committed*
  • all requests with sequence number less than n have been executed
Normal Case Operation

• The algorithm has three phases:
  • *pre-prepare* picks order of requests
  • *prepare* ensures order within views
  • *commit* ensures order across views
• A replica executes a request m if
  • m is *committed*
    • all requests with sequence number less than n have been executed
• Replicas send a reply to the client
  • Client waits for f+1 matching replies
PBFT Agreement Protocol Summary

Network: 3f+1
Quorum: 2f+1
Intersection: f+1

At Most f Malicious Failures
quorum B quorum A

Proposal Validation
Centralized Decentralized Decentralized
Decentralized

Proposal
Decision Making

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
View Change
View Change

- Provide liveness when primary fails
  - Timeouts trigger view changes
- Request a view change
  - send a viewchange request to all
  - new primary requires 2f+1 viewchange messages to accept new role
  - sends new-view with proof (2f+1 viewchange messages)
- View change has a high complexity: $O(n^3)$
Garbage Collection
Garbage Collection

• When to discard messages in the log?
  • periodically checkpoint the state by multicasting CHECKPOINT messages
  • Each node collects $2f+1$ checkpoint messages: proof of correctness
Garbage Collection

- When to discard messages in the log?
  - periodically checkpoint the state by multicasting CHECKPOINT messages
  - Each node collects $2f+1$ checkpoint messages: proof of correctness
Garbage Collection

• When to discard messages in the log?
  • periodically checkpoint the state by multicasting CHECKPOINT messages
  • Each node collects $2f+1$ checkpoint messages: proof of correctness
How to Deal with malicious Failures?
How to Deal with malicious Failures?

• PBFT is an expensive protocol
  • 3 phases of communication
  • 3f+1 nodes
  • $O(n^2)$ message communication ($O(n^3)$ view change)
How to Deal with malicious Failures?

• PBFT is an expensive protocol
  • 3 phases of communication
  • 3f+1 nodes
  • $O(n^2)$ message communication ($O(n^3)$ view change)
• Can an asynchronous protocol perform better?
How to Deal with malicious Failures?

• PBFT is an **expensive** protocol
  • 3 phases of communication
  • 3f+1 nodes
  • $O(n^2)$ message communication ($O(n^3)$ view change)

• Can an asynchronous protocol perform better?
  • **Optimistic Approaches**
    • Execute transactions without ordering [Zyzzyva]
    • Active/passive replication [CheapBFT]
How to Deal with malicious Failures?

- PBFT is an **expensive** protocol
  - 3 phases of communication
  - 3f+1 nodes
  - $O(n^2)$ message communication ($O(n^3)$ view change)

- Can an asynchronous protocol perform better?
  - **Optimistic Approaches**
    - Execute transactions without ordering [*Zyzzyva*]
    - Active/passive replication [*CheapBFT*]
  - Restrict the malicious behavior of nodes
    - Trusted hardware [*MinBFT*][*CheapBFT*]
How to Deal with malicious Failures?

• PBFT is an **expensive** protocol
  • 3 phases of communication
  • 3f+1 nodes
  • \(O(n^2)\) message communication (\(O(n^3)\) view change)

• Can an asynchronous protocol perform better?
  • Optimistic Approaches
    • Execute transactions without ordering [Zyzzyva]
    • Active/passive replication [CheapBFT]
  • Restrict the malicious behavior of nodes
    • Trusted hardware [MinBFT][CheapBFT]
  • Explore a spectrum of performance Trade-off between different complexity metrics
    • Reduce the number of phases (increase the number of nodes) [FaB]
    • Reduce message complexity (increase the number of phases) [HotStuff]
<table>
<thead>
<tr>
<th>Time Model</th>
<th>Fault Model</th>
<th>Consensus Model</th>
<th>Nodes</th>
<th>Phases</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>Crash</td>
<td>Pessimistic</td>
<td>3f+1</td>
<td>1 or 3</td>
<td>O(N)</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>Byzantine</td>
<td>Optimistic</td>
<td>Known</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partially-Synchronous</td>
<td>Hybrid</td>
<td></td>
<td>Unknown</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"Zyzzyva" is the last word in many English-language dictionaries

Zyzzyva: Speculative BFT

• A replica *speculatively* executes a request as soon as it receives a valid pre-prepare message

• Commitment of a request is moved to the client
  • If a request completes at a client, the request will eventually be committed at the server replicas

• Prepare and commit phases are reduced to a single linear phase
  • View change has one more additional phase
Zyzzyva Agreement Protocol: Case 1

Client
replica 0 (Leader)
replica 1
replica 2
replica 3

Request | Order | Reply
Zyzzyva Agreement Protocol: Case 1

Client
replica 0 (Leader)
replica 1
replica 2
replica 3

Request | Order | Reply
Zyzzyva Agreement Protocol: Case 1

Client

Request

Order

Reply

replica 0
(Leader)

replica 1

replica 2

replica 3

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Zyzzyva Agreement Protocol: Case 1

Client

Request | Order | Reply

replica 0
(Leader)

replica 1

replica 2

replica 3

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Zyzzyva Agreement Protocol: Case 1

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Zyzzyva Agreement Protocol: Case 1

- Client receives 3f+1 matching replies
  => all replicas have executed the request in the same total order

![Diagram of the Zyzzyva Agreement Protocol: Case 1](image-url)
## Zyzzyva Agreement Protocol: Case 2

<table>
<thead>
<tr>
<th>Client</th>
<th>Request</th>
<th>Order</th>
<th>Reply</th>
<th>Commit</th>
<th>Local-Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>replica 0 (Leader)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Zyzzyva Agreement Protocol: Case 2

Client

Replica 0
(Leader)

Replica 1

Replica 2

Replica 3

Request | Order | Reply | Commit | Local-Commit

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Zyzzyva Agreement Protocol: Case 2

Client
- Request
- Order
- Reply
- Commit
- Local-Commit

replica 0
(Leader)

replica 1

replica 2

replica 3

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Zyzzyva Agreement Protocol: Case 2

- **Request**
  - Client
  - replica 0 (Leader)
  - replica 1
  - replica 2
  - replica 3

- **Order**
  - replica 0
  - replica 1
  - replica 2
  - replica 3

- **Reply**
  - Receive 2f+1 matching replies
  - Commit

- **Local-Commit**
Zyzzyva Agreement Protocol: Case 2

Commit message contains a commit certificate:
A list of $2f+1$ replica ids and their signed messages
Zyzzyva Agreement Protocol: Case 2

Commit message contains a commit certificate:
A list of $2f+1$ replica ids and their signed messages
Zyzzyva Summary

• One round of message exchange during normal operation

• Impact on view change
  • Need an additional round of message exchange
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td></td>
</tr>
<tr>
<td>Asynchronous</td>
<td></td>
</tr>
<tr>
<td>Partially-Synchronous</td>
<td></td>
</tr>
<tr>
<td>Crash</td>
<td></td>
</tr>
<tr>
<td>Byzantine</td>
<td></td>
</tr>
<tr>
<td>Hybrid</td>
<td></td>
</tr>
<tr>
<td>Pessimistic</td>
<td></td>
</tr>
<tr>
<td>Optimistic</td>
<td></td>
</tr>
<tr>
<td>Known nodes</td>
<td></td>
</tr>
<tr>
<td>Unknown nodes</td>
<td></td>
</tr>
</tbody>
</table>

**HOTSTUFF**


- Linear Communication
- Request Pipelining
- Leader Rotation

**libra**

Facebook

**3f+1 nodes**

- 7 phases
- O(N) Complexity
HotStuff Model

- The same network and quorum size as PBFT
  - 3f+1 nodes in total, Quorums of 2f+1 nodes
HotStuff Model

• The same network and quorum size as PBFT
  • 3f+1 nodes in total, Quorums of 2f+1 nodes

• Linear message complexity
  • Increases the number of phases
  • Each n to n phase of PBFT = an n to 1 + a 1 to n phases of Hotstuff
  • The primary uses (k, n)-threshold signature schema
HotStuff Model

• The same network and quorum size as PBFT
  • 3f+1 nodes in total, Quorums of 2f+1 nodes
• Linear message complexity
  • Increases the number of phases
  • Each $n \text{ to } n$ phase of PBFT = an $n \text{ to } 1$ + a $1 \text{ to } n$ phases of Hotstuff
  • The primary uses $(k, n)$-threshold signature schema

• Leader Rotation
  • A leader is rotated after a single attempt to commit a command/block
  • View-change is part of the normal operation of the system
    • One more phase of communication is needed
    • Linear View change routine
    • PBFT’s View Change has $O(n^3)$ message complexity
## HotStuff Agreement Protocol

<table>
<thead>
<tr>
<th>Client</th>
<th>Request</th>
<th>Prepare</th>
<th>Pre-Commit</th>
<th>Commit</th>
<th>Decide</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>replica 0 (Leader)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HotStuff Agreement Protocol

Client

replica 0
(Leader)
replica 1
replica 2
replica 3

Request  Prepare  Pre-Commit  Commit  Decide  Reply

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
HotStuff Agreement Protocol

Client

replica 0 (Leader)

replica 1

replica 2

replica 3

Request  Prepare  Pre-Commit  Commit  Decide  Reply

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
HotStuff Agreement Protocol

Client

replica 0
(Leader)
replica 1
replica 2
replica 3

Request | Prepare | Pre-Commit | Commit | Decide | Reply
HotStuff Agreement Protocol

Client

<table>
<thead>
<tr>
<th>Replica</th>
<th>Request</th>
<th>Prepare</th>
<th>Pre-Commit</th>
<th>Commit</th>
<th>Decide</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replica 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Leader)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replica 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replica 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replica 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
HotStuff Agreement Protocol

Client

- Request
- Prepare
- Pre-Commit
- Commit
- Decide
- Reply

replica 0 (Leader)

replica 1

replica 2

replica 3

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
HotStuff Agreement Protocol

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
HotStuff Agreement Protocol

Client

replica 0
(Leader)

replica 1

replica 2

replica 3

Request
Prepare
Pre-Commit
Commit
Decide
Reply

Pre-prepare in PBFT
Prepare in PBFT
Commit in PBFT
View Change

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
The Pipeline of HotStuff

clients

replica 0

replica 1

replica 2

replica 3
The Pipeline of HotStuff

clients

replica 0  

replica 1  

replica 2  

replica 3

\[ \text{cmd}_1 \]  

\[ \text{PREPARE} \]  

\[ \text{DECIDE} \]  

\[ \text{COMMIT} \]  

\[ \text{PRE-COMMIT} \]
The Pipeline of HotStuff

clients

replica 0

replica 1

replica 2

replica 3

\( cmd_1 \)  \( cmd_2 \)
The Pipeline of HotStuff

clients

replica 0

replica 1

replica 2

replica 3

$cmd_1$

$cmd_2$

$cmd_3$

PREPARE

PRE-COMMIT

COMMIT

PREPARE

PRE-COMMIT

COMMIT

PRE-COMMIT

COMMIT

DECIDE

DECIDE

DECIDE
The Pipeline of HotStuff

clients

replica 0 $cmd_1$

replica 1

replica 2

replica 3

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
The Pipeline of HotStuff

clients

replica 0

replica 1

replica 2

replica 3
MinBFT


<table>
<thead>
<tr>
<th>synchronous</th>
<th>asynchronous</th>
<th>partially-synchronous</th>
<th>crash</th>
<th>Byzantine</th>
<th>hybrid</th>
<th>Pessimistic</th>
<th>Optimistic</th>
<th>Known nodes</th>
<th>Unknown nodes</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2f+1 nodes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 phases</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>O(N) Complexity</td>
</tr>
</tbody>
</table>
MinBFT

• Uses a tamper proof component: **Unique Sequential Identifier Generator (USIG)**

• All nodes use USIG for message authentication and verification to ensure receiving **symmetric** messages
  • A Byzantine node may decide not to send a message or send it corrupted, but it cannot send two different messages to different replicas

• USIG generates unique identifiers for every message
  • Each identifier is assigned incrementally
  • Each identifier is the successor of the previous one.
MinBFT Agreement Protocol

Requires the same number of replicas, communication phases and message complexity as Paxos
MinBFT Agreement Protocol

Requires the same number of replicas, communication phases and message complexity as Paxos
MinBFT Agreement Protocol

Requires the same number of replicas, communication phases and message complexity as Paxos
MinBFT Agreement Protocol

Requires the same number of replicas, communication phases and message complexity as Paxos
MinBFT Agreement Protocol

Requires the same number of replicas, communication phases and message complexity as Paxos
MinBFT Agreement Protocol

Requires the same number of replicas, communication phases and message complexity as Paxos
CheapBFT

Trusted Hardware
Active/Passive Replication

Synchronous
Asynchronous
Partially-Synchronous

Crash
Byzantine
Hybrid

Pessimistic
Optimistic

Known nodes
Unknown nodes

f+1/2f+1 nodes
2 phases
O(N) Complexity
CheapBFT

Trusted Hardware (called Cash subsystem)
Assigns a unique counter value to each request
Creates Message Certificate and Checks Message Certificate
CASH system can fail only by crashing

Active Passive Replication
$f$ replicas are passive and needed only when there is a failure
CheapBFT Agreement Protocol

1 CheapTiny

- The default protocol, only f+1 replicas participate
- Only f+1 active replicas are selected.
- All the other replicas go in a passive mode
CheapBFT Agreement Protocol

1. CheapTiny
   - The default protocol, only f+1 replicas participate
   - Only f+1 active replicas are selected.
   - All the other replicas go in a passive mode

2. CheapSwitch
   - Switches the protocol from cheapTiny to MinBFT if there is any failure
CheapBFT Agreement Protocol

1. CheapTiny
   - The default protocol, only $f+1$ replicas participate
   - Only $f+1$ active replicas are selected.
   - All the other replicas go in a passive mode

2. CheapSwitch
   - Switches the protocol from cheapTiny to MinBFT if there is any failure

3. MinBFT
   - Involve $2f+1$ active replicas.
   - Eventually, system again switches back to cheapTiny.
CheapTiny Protocol

- **Client**: Request → Prepare → Commit → Reply
- **Active Replicas**: replica 0 (Leader), replica 1
- **Passive Replica**: replica 2

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
CheapTiny Protocol

Client

Request

Prepare

Commit

Reply

replica 0 (Leader)

replica 1

replica 2

Active Replicas

Passive Replica

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
CheapTiny Protocol

Client

<table>
<thead>
<tr>
<th>Request</th>
<th>Prepare</th>
<th>Commit</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>replica 0 (Leader)</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>replica 1</td>
<td>U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Active Replicas

Passive Replica

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
CheapTiny Protocol

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
CheapTiny Protocol

Replica 0 (Leader)
Replica 1
Replica 2

Active Replicas
Passive Replica

Client
Request
Prepare
Commit
Reply

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
CheapSwitch Protocol

Any node can request protocol switch by sending a **PANIC** message to all replicas.

Replicas broadcast the message and wait for Abort History message from the new leader.

New Leader creates and broadcasts an **Abort History**.

Other Replicas validate the abort history and send **Switch** messages to all other replicas.

After receiving $f$ matching switch messages, the history becomes stable.
CheapSwitch Protocol

Panicking Client/Replica

Active Replicas

Passive Replica

replica 0 (Leader)

replica 1

replica 2

Panic History Switch Reply

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
CheapSwitch Protocol

Panicking Client/Replica

Active Replicas
replica 0 (Leader)
replica 1

Passive Replica
replica 2

Panic History Switch Reply

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
CheapSwitch Protocol

Panicking Client/Replica

Active Replicas

Passive Replica

replica 0 (Leader)

replica 1

replica 2

Panic History Switch Reply

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
CheapSwitch Protocol

Panicking Client/Replica

Active Replicas
replica 0 (Leader)
replica 1

Passive Replica
replica 2

Panic History Switch Reply

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
CheapSwitch Protocol

Panicking Client/Replica

Active Replicas
replica 0 (Leader)
replica 1

Passive Replica
replica 2

Panic | History | Switch | Reply

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
What if a network includes both Crash-only and Byzantine nodes?
UpRight Cluster Services

- Hybrid failure model
  - Tolerates both crash and malicious failure
UpRight Cluster Services

• **Hybrid failure model**
  • Tolerates both crash and malicious failure

• **Request quorum**
  • Avoid expensive corner cases with inconsistent client MACs
  • Separate the data path from the control path
UpRight Cluster Services

- **Hybrid failure model**
  - Tolerates both crash and malicious failure

- **Request quorum**
  - Avoid expensive corner cases with inconsistent client MACs
  - Separate the data path from the control path

- **Agreement protocol** is a combination of
  - Zyzzyva's speculative execution
  - Aardvark's techniques for robustness
  - Yin et al.'s techniques for separating agreement and execution
    - While agreement requires 3f+1 nodes, execution needs 2f+1 nodes
UpRight Failure Model

- Tolerate at most \( m \) malicious and at most \( c \) crash faults
  - Quorum: \( 2m + c + 1 \)
  - Intersection: \( m + 1 \)
  - Network: \( 3m + 2c + 1 \)
SeeMoRe

SeeMoRe is derived from Seemorq, a benevolent, mythical bird in Persian mythology which appears as a peacock with the head of a dog and the claws of a lion.


- Synchronous
- Asynchronous
- Partially-Synchronous
- Crash
- Byzantine
- Hybrid
- Pessimistic
- Optimistic
- Known nodes
- Unknown nodes
- 3m+2c+1 nodes
- 2 or 3 phases
- O(N)/O(N²) Complexity

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Hybrid Cloud Environment

Nodes in the private cloud are trusted (crash-only)

Lack of resources to guarantee fault tolerance

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Hybrid Cloud Environment

Nodes in the private cloud are trusted (crash-only)

Lack of resources to guarantee fault tolerance

Nodes in the public cloud are untrusted (Byzantine)

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Hybrid Cloud Environment

Nodes in the private cloud are trusted (crash-only)

Nodes in the public cloud are untrusted (Byzantine)

Can we benefit from both worlds?

Lack of resources to guarantee fault tolerance

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Hybrid Cloud Environment

- Nodes in the private cloud are trusted (crash-only)
- Nodes in the public cloud are untrusted (Byzantine)
- Can we benefit from both worlds?

Lack of resources to guarantee fault tolerance

SeeMoRe

COMPUTER SCIENCE
Ut Santa Barbara
Mode 1: Trusted Primary, Centralized Coordination

- The primary is in the private cloud (Trusted)
- Backups are in both private and public cloud
Mode 1: Trusted Primary, Centralized Coordination

- The primary is in the private cloud (Trusted)
- Backups are in both private and public cloud

Proposal

Primary to backups

Network: 3m+2c+1
Quorum: 2m+c+1
Intersection: m+1

At most \( m \) Malicious
and
At most \( c \) crash faults
Mode 1: Trusted Primary, Centralized Coordination

- The primary is in the private cloud (Trusted)
- Backups are in both private and public cloud

Proposal

Decision Making

Primary to backups

Backups to Primary

Centralized $O(n)$

Network: $3m+2c+1$
Quorum: $2m+c+1$
Intersection: $m+1$

At most $m$ Malicious and
At most $c$ crash faults

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Mode 1: Trusted Primary, Centralized Coordination

- The primary is in the private cloud (Trusted)
- Backups are in both private and public cloud

**Proposal**
- Primary to backups

**Decision Making**
- Backups to Primary

**Centralized O(n)**

**Phases:**
- Two

**Messages:**
- $O(n)$

**Quorum:**
- $2c+m+1$

**Network:**
- $3m+2c+1$

**Quorum:**
- $2m+c+1$

**Intersection:**
- $m+1$

At most **m Malicious**
and
At most **c crash faults**

**Quorum A**

**Quorum B**

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Mode 2: Trusted Primary, Decentralized Coordination

- The primary is still in the private cloud (Trusted)
- The private cloud is not involved in the second phase
- Proxy nodes: $3m+1$ nodes from the public cloud

Goal:
Reduce the load on the private cloud
Mode 2: Trusted Primary, Decentralized Coordination

- The primary is still in the private cloud (Trusted)
- The private cloud is not involved in the second phase
- Proxy nodes: $3m+1$ nodes from the public cloud

**Goal:**
Reduce the load on the private cloud

**Proposal**
Primary to backups

---

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Mode 2: Trusted Primary, Decentralized Coordination

- The primary is still in the private cloud (Trusted)
- The private cloud is not involved in the second phase
- Proxy nodes: $3m+1$ nodes from the public cloud

Proposal

Decision Making

Primary to backups

Proxies to Proxies

Goal:
Reduce the load on the private cloud

Decentralized $O(n^2)$
Mode 2: Trusted Primary, Decentralized Coordination

- The primary is still in the private cloud (Trusted)
- The private cloud is not involved in the second phase
- Proxy nodes: $3m+1$ nodes from the public cloud

**Proposal**
- Primary to backups

**Decision Making**
- Proxies to Proxies

**Phases:**
- Two
- Messages: $O(n^2)$
- Quorum: $2m+1$

**Goal:**
- Reduce the load on the private cloud

---

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
Mode 3: Untrusted Primary, Decentralized Coordination

- The primary is in the public cloud (Untrusted)
- The private cloud is not involved in any phases
- Proxy nodes: $3m+1$ nodes from the public cloud

Goal:
Reduce the load on the private cloud
Reduce latency when there is a large network distance between clouds
Mode 3: Untrusted Primary, Decentralized Coordination

- The primary is in the public cloud (Untrusted)
- The private cloud is not involved in any phases
- Proxy nodes: $3m+1$ nodes from the public cloud

Proposal

Primary to all

Goal:
Reduce the load on the private cloud
Reduce latency when there is a large network distance between clouds
Mode 3: Untrusted Primary, Decentralized Coordination

- The primary is in the public cloud (Untrusted)
- The private cloud is not involved in any phases
- Proxy nodes: $3m+1$ nodes from the public cloud

Goal:
Reduce the load on the private cloud
Reduce latency when there is a large network distance between clouds
Mode 3: Untrusted Primary, Decentralized Coordination

- The primary is in the public cloud (Untrusted)
- The private cloud is not involved in any phases
- Proxy nodes: $3m+1$ nodes from the public cloud

**Proposal**

- Primary to all

**Proposal Validation**

- Proxies to Proxies

**Decision Making**

- Proxies to Proxies

**Goal:**
Reduce the load on the private cloud
Reduce latency when there is a large network distance between clouds
Mode 3: Untrusted Primary, Decentralized Coordination

- The primary is in the public cloud (Untrusted)
- The private cloud is not involved in any phases
- Proxy nodes: $3m+1$ nodes from the public cloud

Goal: Reduce the load on the private cloud
Reduce latency when there is a large network distance between clouds

Phases: Three
Messages: $O(n^2)$
Quorum: $2m+1$
XFT

Partially Synchronous

• Replica p is **partitioned** if p is not in the largest subset of replicas, in which every pair of replicas can communicate among each other within delay $\Delta$.

• replica p is **synchronous** if p is not partitioned
Failures and Anarchy
Failures and Anarchy

- XFT considers three types of failures:
  - c: Number of crash failure
  - m: Number of non-crash (Byzantine) failure
  - p: Number of correct, but partitioned replicas
Failures and Anarchy

• XFT considers three types of failures:
  • c: Number of crash failure
  • m: Number of non-crash (Byzantine) failure
  • p: Number of correct, but partitioned replicas

• Anarchy: The system is in anarchy at a given moment $s$ iff
  • $m(s) > 0$
  • $f = c(s) + m(s) + p(s) > \left\lceil \frac{n-1}{2} \right\rceil$
Failures and Anarchy

• XFT considers three types of failures:
  • c: Number of crash failure
  • m: Number of non-crash (Byzantine) failure
  • p: Number of correct, but partitioned replicas

• Anarchy: The system is in anarchy at a given moment s iff
  • m(s) > 0
  • \( f = c(s) + m(s) + p(s) > \left\lfloor \frac{n-1}{2} \right\rfloor \)

XFT satisfies safety in executions in which the system is never in anarchy
XFT Agreement Protocol (XPaxos)

- Network includes $2f + 1$ replicas where $f$ is network + machine faults
- Uses the active/passive replication technique
- Optimistically replicates requests on only $f+1$ replicas, called a synchronous group
- A view is changed when there is a failure within the synchronous group.
- The view change reconfigures the entire synchronous group
XFT Common Case Protocol

Client

<table>
<thead>
<tr>
<th>Request</th>
<th>Prepare</th>
<th>Commit</th>
<th>Reply</th>
</tr>
</thead>
<tbody>
<tr>
<td>replica 0 (Leader)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Active Replicas

Passive Replicas

replica 3

replica 4

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
XFT Common Case Protocol

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
XFT Common Case Protocol

Client

Active Replicas
replica 0 (Leader)
replica 1
replica 2

Passive Replicas
replica 3
replica 4

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
XFT Common Case Protocol

Client

Active Replicas

Request
Prepare
Commit
Reply

replica 0
(Leader)

replica 1

replica 2

Passive Replicas

replica 3

replica 4

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
XFT Common Case Protocol

Active Replicas

Client
- Request
- Prepare
- Commit
- Reply

replica 0 (Leader)
replica 1
replica 2

Passive Replicas

replica 3
replica 4

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
XFT Common Case Protocol

Request → Prepare → Commit → Reply

Active Replicas:
- replica 0 (Leader)
- replica 1
- replica 2

Passive Replicas:
- replica 3
- replica 4

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE2020
What if the participants are unknown?!
Synchronous
Asynchronous
Partially-Synchronous
Crash
Byzantine
Hybrid
Pessimistic
Optimistic
Known nodes
Unknown nodes
? nodes
1 phases
O(N) Complexity

What is a Blockchain?

- Signed Transactions are grouped into blocks
- Blocks are chained to each other through pointers (Hence blockchain)
- How is the ledger tamper-free?
  Blocks are connected through hash-pointers
What is a Blockchain?

• **Signed** Transactions are grouped into blocks
• Blocks are chained to each other through pointers (Hence blockchain)
• **How is the ledger tamper-free?**
  Blocks are connected through **hash-pointers**

![Diagram of a blockchain with transactions](image-url)

TX₁
TX₂
·
·
TXₙ
What is a Blockchain?

• **Signed** Transactions are grouped into blocks
• Blocks are chained to each other through pointers (Hence blockchain)
• How is the ledger tamper-free?
  
  Blocks are connected through **hash-pointers**
What is a Blockchain?

- **Signed** Transactions are grouped into blocks
- Blocks are chained to each other through pointers (Hence blockchain)
- **How is the ledger tamper-free?**
  Blocks are connected through **hash-pointers**
What is a Blockchain?

- **Signed** Transactions are grouped into blocks
- Blocks are chained to each other through pointers (Hence blockchain)
- **How is the ledger tamper-free?**
  Blocks are connected through **hash-pointers**
What is a Blockchain?

- Signed Transactions are grouped into blocks
- Blocks are chained to each other through pointers (Hence blockchain)
- How is the ledger tamper-free?
  Blocks are connected through hash-pointers
What is a Blockchain?

- **Signed** Transactions are grouped into blocks
- Blocks are chained to each other through pointers (Hence blockchain)
- How is the ledger tamper-free?
  - Blocks are connected through hash-pointers

![Diagram of blockchain structure]
Making Progress

Hash()  
TX₁  
TX₂  
.  
.  
TXₙ

Hash()  
TX₁  
TX₂  
.  
.  
TXₙ

Hash()  
TX₁  
TX₂  
.  
.  
TXₙ

............

Hash()  
TX₁  
TX₂  
.  
.  
TXₙ

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
Making Progress

• To make progress:
  • Network nodes **validate** new transactions are consistent.
Making Progress

• To make progress:
  • Network nodes **validate** new transactions are consistent.
  • Network nodes need to agree on the next block to be added to the blockchain
Making Progress

• To make progress:
  • Network nodes **validate** new transactions are consistent.
  • Network nodes need to agree on the next block to be added to the blockchain.

Consensus
Permissionless Blockchains have **Unknown Number** of Participants
Reach Consensus Using Mining

Permissionless Blockchains have Unknown Number of Participants
Reach Consensus Using Mining
Replace Communication with Computation!!

Permissionless Blockchains have Unknown Number of Participants
Proof of Work Consensus

• Intuitively, network nodes race to solve a puzzle
• This puzzle is computationally expensive
• Once a network node finds (mines) a solution:
  • It adds its block of transactions to the blockchain
  • It multi-casts the solution to other network nodes
  • Other network nodes accept and verify the solution
Mining Details
Mining Details
Mining Details
Mining Details

TX₁
TX₂
... 
TXₙ

TX₁
TX₂
... 
TXₙ
Mining Details

TX₁
TX₂
TXₙ

TX₁
TX₂
TXₙ

TX₁
TX₂
TXₙ

TX₁
TX₂
TXₙ
Mining Details

TX_1 → TX_2 → ... → TX_n
Mining Details

TX_1 → TX_2 → ... → TX_n

TX_1
TX_2
...
TX_n
Mining Details

TX₁
TX₂
TXₙ

TX₁
TX₂
TXₙ

TX₁
TX₂
TXₙ
Mining Details

TX_1
TX_2
\ldots
TX_n

TX_1
TX_2
\ldots
TX_n

TX_1
TX_2
\ldots
TX_n

40 Years of Consensus: Amini, Agrawal, El Abbadi, ICDE2020
Mining Details

\[ TX_1 \]
\[ TX_2 \]
\[ \ldots \]
\[ TX_n \]

TX\text{\_reward}

TX\text{\_reward}

TX\text{\_reward}
Mining Details

TX \_1
TX \_2
\ldots
TX \_n

TX \_1
TX \_2
\ldots
TX \_n

TX \_1
TX \_2
\ldots
TX \_n

TX\_reward
Mining Details
Mining Details

<table>
<thead>
<tr>
<th>Version</th>
<th>Previous Block Hash</th>
<th>Merkle Tree Root Hash</th>
<th>Time Stamp</th>
<th>Current Target Bits</th>
<th>Nonce</th>
<th>TX\text{\textunderscore reward}</th>
<th>TX\text{\textunderscore 1}</th>
<th>TX\text{\textunderscore 2}</th>
<th>\ldots</th>
<th>TX\text{\textunderscore n}</th>
<th>TX\text{\textunderscore 1}</th>
<th>TX\text{\textunderscore 2}</th>
<th>\ldots</th>
<th>TX\text{\textunderscore n}</th>
</tr>
</thead>
</table>

40 Years of Consensus, Afzali, Agrawal, El Abbadi, ICDE2020
Mining Details

SHA256(TX\text{reward}) < D

- Version
- Previous Block Hash
- Merkle Tree Root Hash
- Time Stamp
- Current Target Bits
- Nonce

Transactions

TX_1
TX_2
\ldots
TX_n

TX_1
TX_2
\ldots
TX_n

TX_1
TX_2
\ldots
TX_n

TX_1
TX_2
\ldots
TX_n

40 Years of Consensus, Amini, Agrawal, El Abbadi, ICDE2020
Mining Details

Transactions

Version
Previous Block Hash
Merkle Tree Root Hash
Time Stamp
Current Target Bits
Nonce

SHA256(TX_{reward}) < D
Mining Details

- $\text{TX}_{\text{reward}}$ is self signed (also called coinbase transaction)
- $\text{TX}_{\text{reward}}$ is bitcoin’s way to create new coins
- The reward value is halved every 4 years (210,000 blocks)
- Currently, it’s 12.5 Bitcoins per block
- Incentives network nodes to mine

\[ \text{SHA256}(\phantom{\text{TX}_{\text{reward}}}) < D \]

<table>
<thead>
<tr>
<th>Header</th>
<th>Transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>$\text{TX}_{\text{reward}}$</td>
</tr>
<tr>
<td>Previous Block Hash</td>
<td>$\text{TX}_1$</td>
</tr>
<tr>
<td>Merkle Tree Root Hash</td>
<td>$\text{TX}_2$</td>
</tr>
<tr>
<td>Time Stamp</td>
<td>$\text{TX}_n$</td>
</tr>
<tr>
<td>Current Target Bits</td>
<td></td>
</tr>
<tr>
<td>Nonce</td>
<td>$\text{TX}_{\text{reward}}$</td>
</tr>
<tr>
<td></td>
<td>$\text{TX}_1$</td>
</tr>
<tr>
<td></td>
<td>$\text{TX}_2$</td>
</tr>
<tr>
<td></td>
<td>$\text{TX}_n$</td>
</tr>
</tbody>
</table>
Mining Details

TX1
TX2
TXn

TX1
TX2
TXn

TX1
TX2
TXn

TX1
TX2
TXn

TXreward

SHA256(

Transactions

Header)

Version
Previous Block Hash
Merkle Tree Root Hash
Time Stamp
Current Target Bits
Nonce

TXreward
TX1
TX2
TXn
## Mining Details

### Transactions

<table>
<thead>
<tr>
<th>TX\text{reward}</th>
<th>TX\text{1}</th>
<th>TX\text{2}</th>
<th>...</th>
<th>TX\text{n}</th>
</tr>
</thead>
</table>

### Header

- **Version**
- **Previous Block Hash**
- **Merkle Tree Root Hash**
- **Time Stamp**
- **Current Target Bits**
- **Nonce**

SHA256( ) \leq D

---

40 Years of Consensus, Amini, Agrawal, El Abbadi, ICDE2020
Mining Details

- D: dynamically adjusted difficulty
- Difficulty is adjusted every 2016 blocks (almost 2 weeks)

SHA256( ) < D

Transactions

- Version
- Previous Block Hash
- Merkle Tree Root Hash
- Time Stamp
- Current Target Bits
- Nonce

40 Years of Consensus, Amini, Agrawal, El Abbadi, ICDE2020
Mining Details

- D: dynamically adjusted difficulty
  - 256 bits
- Difficulty is adjusted every 2016 blocks (almost 2 weeks)

SHA256( ) < \( D \)

Version
Previous Block Hash
Merkle Tree Root Hash
Time Stamp
Current Target Bits
Nonce

Transactions

\( TX_{reward} \)

\( TX_1 \)
\( TX_2 \)
\( \cdots \)
\( TX_n \)

\( TX_1 \)
\( TX_2 \)
\( \cdots \)
\( TX_n \)

\( TX_1 \)
\( TX_2 \)
\( \cdots \)
\( TX_n \)

\( TX_1 \)
\( TX_2 \)
\( \cdots \)
\( TX_n \)

\( TX_1 \)
\( TX_2 \)
\( \cdots \)
\( TX_n \)
Mining Details

- D: dynamically adjusted difficulty
- Difficulty is adjusted every 2016 blocks (almost 2 weeks)

Difficulty bits

Version
Previous Block Hash
Merkle Tree Root Hash
Time Stamp
Current Target Bits
Nonce

Transactions

TX1
TX2
... 
TXn

TX1
TX2
... 
TXn

TX1
TX2
... 
TXn

TX1
TX2
... 
TXn

TXrward

SHA256( ) < D

• D: dynamically adjusted difficulty

256 bits

• Difficulty is adjusted every 2016 blocks (almost 2 weeks)

40 Years of Consensus, Amin, Agrawal, El Abbadi, ICDE2020
Mining Details: Block Contents

TX_1
TX_2
·
·
TX_n

TX_1
TX_2
·
·
TX_n

TX_1
TX_2
·
·
TX_n
Mining Details: Block Contents

Transactions

TX_{\text{reward}}
TX_1
TX_2
\cdots
TX_n
Mining Details: Block Contents

\[
\text{SHA256} (\text{TX}_1, \text{TX}_2, \ldots, \text{TX}_n) < D
\]

Transactions

\[
\text{TX}_{\text{reward}}, \text{TX}_1, \text{TX}_2, \ldots, \text{TX}_n
\]
Mining Details: Block Contents

Transactions

SHA256(TX_1, TX_2, ..., TX_n, TX_reward) < D

Amiri, Agrawal, El Abbadi, ICDE2020
Mining Details: Block Contents

- D: dynamically adjusted difficulty

- Difficulty is adjusted every 2016 blocks (almost 2 weeks)

\[
\text{SHA256}(\text{Transactions}) < D
\]
Mining Details: Block Contents

- D: dynamically adjusted difficulty
  - 256 bits

- Difficulty is adjusted every 2016 blocks (almost 2 weeks)

\[ \text{SHA256(} \quad \text{Transactions} \quad \text{)} < D \]

- \( TX_{\text{reward}} \)
- \( TX_1 \)
- \( TX_2 \)
  - ...
- \( TX_n \)
Mining Details: Block Contents

- D: dynamically adjusted difficulty
  - 256 bits
- Difficulty is adjusted every 2016 blocks (almost 2 weeks)

\[
\text{Transactions} \rightarrow \text{SHA256(} \sum_{i=1}^{n} \text{TX}_i \text{)} \rightarrow D
\]

Transactions

\[
\text{TX}_{\text{reward}}
\]

\[
\text{TX}_1
\]

\[
\text{TX}_2
\]

\[
\text{TX}_n
\]

40 Years of Consensus - Amin, Agrawal, El Abbadi, ICDE2020
Mining Details

• Find a nonce that results in SHA256(block) < Difficulty
Mining Details

• Find a **nonce** that results in SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>02000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 A9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>TX_{\text{reward}}</td>
</tr>
<tr>
<td></td>
<td>TX_1</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>TX_n</td>
</tr>
</tbody>
</table>

40 Years of Consensus- Amin, Agrawal, El Abbadi, ICDE2020
Mining Details

- Find a **nonce** that results in SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>0200000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>□</td>
</tr>
</tbody>
</table>

Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)

```
000000000000000000CF3620D570D08D1799A1CAFBBFAE512FD82124665ECA0
```

18 zeros

40 Years of Consensus- Amini, Agrawal, El Abbadi, ICDE2020
Mining Details

- Find a **nonce** that results in SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>02000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24</td>
</tr>
<tr>
<td></td>
<td>AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA</td>
</tr>
<tr>
<td></td>
<td>145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>TX\textsubscript{reward}</td>
</tr>
<tr>
<td></td>
<td>TX\textsubscript{1}</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>TX\textsubscript{n}</td>
</tr>
</tbody>
</table>

Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)

\[
000000000000000000cf3620d570d08d1799a1cafbbae512fdba2124665eca0
\]

18 zeros

SHA256(V,P,M,T,C,0) =

\[
BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B032429C72
\]
Mining Details

• Find a **nonce** that results in $\text{SHA256}($block$) < \text{Difficulty}$

<table>
<thead>
<tr>
<th>Version (4B)</th>
<th>02000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\text{TX}_{\text{reward}}$</td>
</tr>
<tr>
<td></td>
<td>$\text{TX}_1$</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>$\text{TX}_n$</td>
</tr>
</tbody>
</table>

**Difficulty** is a function of Current Target Bits (Largest possible Target/Current Target)

```
000000000000000000cf3620d570d08d1799a1cafbbfae512fdba2124665ec00
```

18 zeros

**SHA256($V,P,M,T,C,0$)** =

```
BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B0324
```
Mining Details

- Find a **nonce** that results in SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Version (4B)</th>
<th>02000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>TX_reward, TX_1, ..., TX_n</td>
</tr>
</tbody>
</table>

Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)

```
000000000000000000cf3620d570d08d1799a1cafbbfae512fdb2124665eca0
```

18 zeros

SHA256(V, P, M, T, C, 0) = BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B0324
SHA256(V, P, M, T, C, 1) = DF64342507E785FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872A45D8

40 Years of Consensus- Amin, Agrawal, El Abbadi, ICDE2020
Mining Details

- Find a **nonce** that results in SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Version (4B)</th>
<th>02000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td></td>
</tr>
<tr>
<td>TX_reward</td>
<td></td>
</tr>
<tr>
<td>TX_1</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>TX_n</td>
<td></td>
</tr>
</tbody>
</table>

Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)

000000000000000000cf3620d570d08d1799a1cafbbfae512fdba2124665eca0

18 zeros

SHA256(V,P,M,T,C,0) = BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B0324
SHA256(V,P,M,T,C,1) = DF64342507E785FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872
# Mining Details

- **Find a nonce** that results in $\text{SHA256}(\text{block}) < \text{Difficulty}$

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>02000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368D6FC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>$\text{TX}_{\text{reward}}$ $\text{TX}_1$ $\text{TX}_n$</td>
</tr>
</tbody>
</table>
Mining Details

- Find a **nonce** that results in SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>02000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>TX_reward</td>
</tr>
<tr>
<td></td>
<td>TX_1</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>TX_n</td>
</tr>
</tbody>
</table>

Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)

- SHA256(V,P,M,T,C,0) = BD72804EE251889F9013C100767999B57E92EC5B6ADDBEF64F2DF1B0324
- SHA256(V,P,M,T,C,1) = DF64342507E78FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872
- SHA256(V,P,M,T,C,2) = 0000000CC7F94221B95F4E606E037D31C10417435DEE60A61C627B64324

40 Years of Consensus - Amini, Agrawal, El Abbadi, ICDE2020
Mining Details

- Find a **nonce** that results in $\text{SHA256}(\text{block}) < \text{Difficulty}$

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>02000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>$\text{TX}_{\text{reward}}$</td>
</tr>
<tr>
<td></td>
<td>$\text{TX}_1$</td>
</tr>
<tr>
<td></td>
<td>$\text{...}$</td>
</tr>
<tr>
<td></td>
<td>$\text{TX}_n$</td>
</tr>
</tbody>
</table>

**Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)**

- SHA256($V,P,M,T,C,0$) = BD72804EE251889F9013C100767999B57E92EC5B6ADBDF64F2DF1B0324
- SHA256($V,P,M,T,C,1$) = DF64342507E785FDC0D4776D7142BB2BC6467F09E0040A3E9F65E38872
- SHA256($V,P,M,T,C,2$) = 0000000CC7F94221B95F4E606E037D31C10417435DEE60A61C627B64324

- SHA256($V,P,M,T,C,01F04A1C$) = 0000000000000001E3BFE56AD29732B81128B79356442C8B87F6CED8B6610

- SHA256($V,P,M,T,C,18$ zeros) = 000000000000000000cf3620d570d08d1799a1cafbbfae512fdb4e65ec0
Mining Details

• Find a **nonce** that results in SHA256(block) < Difficulty

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version (4B)</td>
<td>02000000</td>
</tr>
<tr>
<td>Previous Block Hash (32B)</td>
<td>25F947B7C18A1E4E2DF96D0D4368DFC24 AA9C4EC8C3D6B51A4C4935409D58FED</td>
</tr>
<tr>
<td>Merkle Tree Root Hash (32B)</td>
<td>4E04D109A3A7A0460AD2DFD95A4F0FAA 145F3249BEE9F371F8204D16C01D4921</td>
</tr>
<tr>
<td>Time Stamp (4B)</td>
<td>5C9F3E20</td>
</tr>
<tr>
<td>Current Target Bits (4B)</td>
<td>172E6117</td>
</tr>
<tr>
<td>Nonce (4B)</td>
<td>TX_reward TX_1 \cdot \cdot TX_n</td>
</tr>
<tr>
<td>Difficulty</td>
<td>00000000000000000000000000000000cf3620d570d08d1799a1cafbbfae512fdba2124665eca0</td>
</tr>
</tbody>
</table>

Difficulty is a function of Current Target Bits (Largest possible Target/Current Target)

- SHA256(V,P,M,T,C,0) = BD72804EE251889F9013C100767999B57E92EC5B6ADBDBF64F2DF1B0324
- SHA256(V,P,M,T,C,1) = DF64342507E785FDC0D4C776D7142BB2BC6467F09E0040A3E9F65E38872
- SHA256(V,P,M,T,C,2) = 00000000CC7F94221B95F4E6060E037D31C10417435DEE60A61C627B64324

- SHA256(V,P,M,T,C,01F04A1C) = 00000000000000001E3BFE56AD29732B81128B79356442C88B7F6CED8

- 18 zeros
- 7 zeros
- 18 zeros
Bitcoin Forks

• Mining is probabilistic ⇨ Forks! Aborts!
Bitcoin Forks

• Mining is probabilistic → Forks! Aborts!
Bitcoin Forks

- Mining is probabilistic → Forks! Aborts!

- Transactions in the forked blocks might have conflicts
- Forks have to be eliminated
Bitcoin Forks

- Mining is probabilistic $\rightarrow$ Forks! Aborts!
Bitcoin Forks

• Mining is probabilistic $\rightarrow$ Forks! Aborts!
Bitcoin Forks

• Mining is probabilistic → Forks! Aborts!
Bitcoin Forks

• Mining is probabilistic \(\rightarrow\) Forks! Aborts!

• Miners join the longest chain to resolve forks
Bitcoin Forks

• Mining is probabilistic → Forks! Aborts!
Bitcoin Forks

• Mining is probabilistic → Forks! Aborts!

• Transactions in this block are aborted/resubmitted
Mining Big Picture
Mining Big Picture
Mining Big Picture
Mining Big Picture

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
First Issue: Mining Centralization

- Chinese pools control ~81% of the network hash rate
  - China: 81%
  - Czech Republic: 10%
  - Iceland: 2%
  - Japan: 2%
  - Georgia: 2%
  - Russia: 1%
Second Issue: PoW consumes lots of electricity

- The amount of electricity used annually by the Bitcoin network could satisfy the energy needs of the University of Cambridge for 442 years.
- The amount of electricity consumed every year by always-on but inactive home devices in the USA alone could power the Bitcoin network for 2.8 years.
- The amount of electricity consumed by the Bitcoin network in one year could power all tea kettles used to boil water for 17 years.
- The European Union could power the Bitcoin network for 2.6 years.

40 Years of Consensus - Amiri, Agrawal, El Abbadi, ICDE 2020
Other Issues

- Weak finality guarantees
- Selfish mining and other attacks
- Suboptimal light client support
Similar to PoS but the difference is that it depends on various other factors called weights.

When a user initiates a transaction, miners or supercomputers try to solve a problem or puzzle to verify it.

A user is encouraged to spend more until he/she becomes a validator to create a block.

Users send the coins back into their wallet that they can’t recover from will get rewards based on the amount.

Using this protocol you can utilize the capacity or storage space of users’ hard drive.

Users will be able to make customized tokens and use it on their farms for better security.

Similar to PoW but the difference is that it focuses more on consumption.

OMGs don’t have blockchain data structure and can handle transactions asynchronously.

Focuses on a gamified way of a block verification among the professional node controllers.

A single validator can bundle proposed transactions and create a new block.

Same as PoS but users with more coins will get to vote and elect witnesses.

Users that frequently send and receive transactions will get paid for that.

Uses both PoS and PoW to ensure the reward points are on time.
PROOF OF STAKE
Proof of Stake

• A stakeholder who has $p$ fraction of the coins in circulation creates a new block with $p$ probability

• Don’t the rich get richer?
  • Randomized block selection
    • Combination of a random number and the stake size
  • Coin age-based selection
    • The number of coins * the number of days the coins have been held.
    • Coins that have been unspent for at least 30 days begin competing for the next block.
    • Older and larger sets of coins have a greater probability of signing the next block.
    • The probability of finding the next block reaches a maximum after 90 days
Tendermint

- Has its own consensus protocol
- Extends PBFT with leader rotation

LibraBFT
- A variant of HotStuff

MultiChain
- Uses PBFT
- Incorporates leader rotation

Libra

40 Years of Consensus- Amiri, Agrawal, El Abbadi, ICDE2020
THANK YOU!
Questions?