Mencius: Building Efficient Replicated State Machines for WANs

Y. Mao, F. Junqueira, K. Marzullo
Presented by Tony Zhang
State Machine Replication
The Wide Area Model

~100 ms
Why not Paxos?

1. Unbalanced Communication
Why not Paxos?

1. Unbalanced Communication
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

2. High End-to-end Latency
Why not Paxos?

3. Computational Bottleneck at Leader

- Every client request is ordered by the leader
- Leader processes $O(n)$ messages, but $O(1)$ for non-leader replicas
- Throughput is bounded by max leader CPU utilization
Outline

- Motivation
- Protocol
- Optimizations
- Evaluation
Mencius
Mencius

Multi-Leader protocol
Mencius

- Multi-Leader protocol
- Partition consensus instances among \( n \) replicas.
Mencius

- Multi-Leader protocol

- Partition consensus instances among $n$ replicas.

- E.g. Each server $i$ is assigned slots $i$, $i + n$, $i + 2n$, ...
Mencius

- Multi-Leader protocol
- Partition consensus instances among $n$ replicas.
  - E.g. Each server $i$ is assigned slots $i$, $i + n$, $i + 2n$, ...
- Each server coordinates its own slots.
Mencius

- Multi-Leader protocol

- Partition consensus instances among \( n \) replicas.

- E.g. Each server \( i \) is assigned slots \( i, i + n, i + 2n, \ldots \)

- Each server coordinates its own slots.

You are a leader! You are a leader!
Assumptions
Assumptions

- Crash failure model
Assumptions

- Crash failure model
- Unreliable failure detector
Assumptions

- Crash failure model
- Unreliable failure detector
- Asynchronous FIFO communication channels
Assumptions

- Crash failure model
- Unreliable failure detector
- Asynchronous FIFO communication channels
- E.g. TCP
Simple Consensus

- Special no-op value
- For each slot, only coordinator can propose any value
Simple Consensus

- Special no-op value

- For each slot, only coordinator can propose any value

```
1 2 3 4 5 ...
```

Simple Consensus
Simple Consensus

- Special no-op value
- For each slot, only coordinator can propose any value

Benefit: Servers can learn no-op without majority agreement!
Coordinated Paxos

**SUGGEST**
- Coordinator proposes value \( v \) in round \( r \)

**SKIP**
- Coordinator skips its turn by proposing no-op in round \( r \)

**REVOKE**
- If process \( p \) suspects that the coordinator for slot \( s \) failed, it revokes the right of the coordinator to propose a value for \( s \), by running a view change.
Example of Coordinated Paxos

$p_0$ coordinator of instances 0, 3, 6, ________________

$p_1$ coordinator of instances 1, 4, 7, ________________

$p_2$ coordinator of instances 2, 5, 8, ________________

Slot 0
Example of Coordinated Paxos

- $p_0$ coordinator of instances 0, 3, 6,
- $p_1$ coordinator of instances 1, 4, 7,
- $p_2$ coordinator of instances 2, 5, 8,

$p_0$ propose $x$

sugg

Slot 0
Example of Coordinated Paxos

$p_0$ coordinator of instances 0, 3, 6,

$p_1$ coordinator of instances 1, 4, 7,

$p_2$ coordinator of instances 2, 5, 8,

$p_0$ propose $x$

sugg

ack

Slot 0
Example of Coordinated Paxos

$p_0$ coordinator of instances 0, 3, 6,

$p_1$ coordinator of instances 1, 4, 7,

$p_2$ coordinator of instances 2, 5, 8,
Example of Coordinated Paxos

$p_0$ coordinator of instances 0, 3, 6,

$p_1$ coordinator of instances 1, 4, 7,

$p_2$ coordinator of instances 2, 5, 8,

Slot 0
Example of Coordinated Paxos

\[ p_0 \text{ coordinator of instances } 0, 3, 6, \]
\[ p_1 \text{ coordinator of instances } 1, 4, 7, \]
\[ p_2 \text{ coordinator of instances } 2, 5, 8, \]

Slot 0

\[ p_0 \text{ propose } x \]
\[ p_0 \text{ learn } x \]

Slot 1

\[ \text{skip} \]
\[ p_1 \text{ skips} \]
Example of Coordinated Paxos

$p_0$ coordinator of instances 0, 3, 6,

$p_1$ coordinator of instances 1, 4, 7,

$p_2$ coordinator of instances 2, 5, 8,
Example of Coordinated Paxos

$p_0$ coordinator of instances 0, 3, 6,

$p_1$ coordinator of instances 1, 4, 7,

$p_2$ coordinator of instances 2, 5, 8,
Example of Coordinated Paxos

$p_0$ coordinator of instances 0, 3, 6,

$p_1$ coordinator of instances 1, 4, 7,

$p_2$ coordinator of instances 2, 5, 8,
Example of Coordinated Paxos

$p_0$ coordinator of instances 0, 3, 6,

$p_1$ coordinator of instances 1, 4, 7,

$p_2$ coordinator of instances 2, 5, 8,

$p_0$ propose $x$

$p_0$ learn $x$

Slot 0

$p_0$ learns $p_1$ skips

$p_0$ revokes $p_2$

$p_0$ learns $p_2$

Slot 1

$p_0$ skips

$p_0$ learns

Slot 2

$p_0$ skips

$p_0$ learns

$p_0$ propose $x$

$p_0$ learn $x$
Example of Coordinated Paxos

$p_0$ coordinator of instances 0, 3, 6,

$p_1$ coordinator of instances 1, 4, 7,

$p_2$ coordinator of instances 2, 5, 8,
Example of Coordinated Paxos

$p_0$ coordinator of instances 0, 3, 6,

$p_1$ coordinator of instances 1, 4, 7,

$p_2$ coordinator of instances 2, 5, 8,
Example of Coordinated Paxos

$p_0$ coordinator of instances 0, 3, 6,

$p_1$ coordinator of instances 1, 4, 7,

$p_2$ coordinator of instances 2, 5, 8,

$p_0$ propose $x$

$p_0$ learn $x$

$p_0$ propose $x$

$p_0$ revokes $p_2$

$p_0$ learn noop
The Protocol
Rule 1:

- Each server $p$ maintains its next Simple Consensus slot number $I_p$
- Update $I_p$ upon client request
The Protocol

Rule 1:

- Each server $p$ maintains its next Simple Consensus slot number $I_p$
- Update $I_p$ upon client request

Rule 2: If $p$ receives SUGGEST for slot $i > I_p$, then $p$

- Update $I_p' :=$ smallest slot $> i$ that $p$ coordinates
- Execute SKIP actions for range $[I_p, I_p')$
The Protocol

Rule 1:

- Each server $p$ maintains its next Simple Consensus slot number $I_p$
- Update $I_p$ upon client request

Rule 2: If $p$ receives SUGGEST for slot $i > I_p$, then $p$

- Update $I'_p :=$ smallest slot $> i$ that $p$ coordinates
- Execute SKIP actions for range $[I_p, I'_p)$

Rule 3: If $p$ suspects that $q$ has failed, and $C_q$ is smallest slot coordinated by $q$ not learned by $p$, then

- $p$ revokes $q$ for all slots in range $[C_q, I_p)$ that $q$ coordinates
The Protocol

Rule 1:

- Each server $p$ maintains its next Simple Consensus slot number $I_p$
- Update $I_p$ upon client request

Rule 2: If $p$ receives SUGGEST for slot $i > I_p$, then $p$

- Update $I'_p :=$ smallest slot $> i$ that $p$ coordinates
- Execute SKIP actions for range $[I_p, I'_p)$

Rule 3: If $p$ suspects that $q$ has failed, and $C_q$ is smallest slot coordinated by $q$ not learned by $p$, then

- $p$ revokes $q$ for all slots in range $[C_q, I_p)$ that $q$ coordinates

Rule 4: If $p$ suggests $\nu \neq \text{no-op}$ and learns that no-op is chosen, then

- $p$ suggests $\nu$ again
Outline

Motivation

Protocol

Optimizations

Evaluation
Optimizations
Suppose server $q$ broadcasts SUGGEST
Optimizations

Suppose server $q$ broadcasts SUGGEST

Op 1: When $p$ receives SUGGEST and $p$ updates $I_p$, $p$ need not send SKIP messages to $q$. Piggyback info on ACCEPT reply.
Suppose server $q$ broadcasts SUGGEST

Op 1: When $p$ receives SUGGEST and $p$ updates $I_p$, $p$ need not send SKIP messages to $q$. Piggyback info on ACCEPT reply.

Op 2: For $r \neq q$, server $p$ need not send SKIP messages to $r$. Piggyback info on next SUGGEST.
Optimizations

Suppose server $q$ broadcasts SUGGEST

Op 1: When $p$ receives SUGGEST and $p$ updates $I_p$, $p$ need not send SKIP messages to $q$. Piggyback info on ACCEPT reply.

Op 2: For $r \neq q$, server $p$ need not send SKIP messages to $r$. Piggyback info on next SUGGEST.

Acc 1: Server $p$ propagates SKIP messages to $r$ if total number of outstanding SKIP msgs $> \alpha$, or deferred for time $> \tau$
Optimizations

Suppose server $q$ broadcasts SUGGEST

Op 1: When $p$ receives SUGGEST and $p$ updates $I_p$, $p$ need not send SKIP messages to $q$. Piggyback info on ACCEPT reply.

Op 2: For $r \neq q$, server $p$ need not send SKIP messages to $r$. Piggyback info on next SUGGEST.

Acc 1: Server $p$ propagates SKIP messages to $r$ if total number of outstanding SKIP msgs $> \alpha$, or deferred for time $> \tau$

Op 3: Suppose $p$ suspects that $q$ failed, and $C_q$ is smallest instance coordinated by $q$ not learned by $p$. Then
Optimizations

Suppose server $q$ broadcasts SUGGEST

Op 1: When $p$ receives SUGGEST and $p$ updates $I_p$, $p$ need not send SKIP messages to $q$. Piggyback info on ACCEPT reply.

Op 2: For $r \neq q$, server $p$ need not send SKIP messages to $r$. Piggyback info on next SUGGEST.

Acc 1: Server $p$ propagates SKIP messages to $r$ if total number of outstanding SKIP msgs $> \alpha$, or deferred for time $> \tau$

Op 3: Suppose $p$ suspects that $q$ failed, and $C_q$ is smallest instance coordinated by $q$ not learned by $p$. Then

$p$ revokes $q$ for all instances in range $[C_q, I_p + 2\beta)$ that $q$ coordinates if $C_q < I_p + \beta$
Evaluation

- Read/write register service with $\kappa$ registers
  - 1 bit op type
  - 2 byte register name
  - 4 byte request id
  - $\rho$ byte dummy payload
  - 50% read-write ratio
Latency-Throughput

\(\rho\) is payload size in bytes

(a) \(\rho = 4,000\), no network variance

(b) \(\rho = 0\), no network variance

(c) \(\rho = 0\), with network variance
Mencius has better end-to-end latency than Paxos

Latency Throughput

(a) $\rho = 4,000$, no network variance

(b) $\rho = 0$, no network variance

(c) $\rho = 0$, with network variance

$\rho$ is payload size in bytes
Throughput Under Failure

(a) One Mencius server crashes
(b) Paxos leader crashes
(c) One Paxos non-leader crashes
Throughput Under Failure

(a) One Mencius server crashes

(b) Paxos leader crashes

(c) One Paxos non-leader crashes
Throughput Under Failure

(a) One Mencius server crashes
(b) Paxos leader crashes
(c) One Paxos non-leader crashes
Throughput Under Failure

Mencius has reduced performance when any replica fails. Tragic 🙁
Throughput Under Failure

Throughput (ops)

(a) One Mencius server crashes
(b) Paxos leader crashes
(c) One Paxos non-leader crashes

Mencius has reduced performance when any replica fails. Tragic 😞

Paxos has reduced performance only when leader fails.
Scalability

Network bound
Scalability

Network bound

CPU bound
Discussion

Mencius is a multi-leader protocol designed for WAN

Compared to Paxos

- Better wide area latency ✔
- Better peak throughput ✔
- Better scalability ✔
- Performance degradation on failure ❌
Cheers!