Mencius: Another Paxos Variant???

Authors: Yanhua Mao, Flavio P. Junqueira, Keith Marzullo

Presented by Isaiah Mayerchak & Yijia Liu
State Machine Replication in WANs

- **WAN = Wide Area Network**
  - Web services, grid services, etc
  - Higher latency, less bandwidth than LANs

- **Goals:**
  - High throughput for high client load
  - Low latency for low client load

- **2 possible bottlenecks for throughput:**
  - Network-bound system
  - CPU-bound system
State Machine Replication in WANs

- **Paxos**
  - Simplicity => high throughput
  - Single leader causes issues:
    - CPU bottleneck
    - Unbalanced communication
    - High latency for far-off clients

- **Fast Paxos & CoReFP**
  - No single leader => no bottleneck
  - Higher message complexity => lower throughput
Another Greek island to the rescue?
Another Greek island to the rescue?

Mencius (Meng Zi)
Mencius: Everyone is a leader!

- Simple consensus => Coordinated Paxos
- Optimizations to reduce # of sent messages => Mencius
- Achieves high throughput & low latency especially when no failures are present
Deriving Mencius: Assumptions

- **Crash process failure:**
  - servers fail by crashing
  - can recover later from stable storage

- **Unreliable failure detector:**
  - eventually, all faulty servers and only faulty servers are suspected
  - temporary false suspicion is allowed

- **Asynchronous FIFO communication channel:**
  - TCP FIFO channels
  - messages between two correct servers are eventually delivered
  - recovery mechanism that does not depend on FIFO channels
Deriving Mencius: Simple consensus

● **Definition:**
  ○ Simple consensus is consensus in which the values a server can propose are restricted

● **Message types:**
  ○ \( V \)
  ○ \( \text{No-op} \)

● **Server types:**
  ○ Coordinator (a.k.a. leader): one special server
  ○ The other servers

● **Restrictions:**
  ○ The coordinator can propose any command (\( V \) and \( \text{no-op} \))
  ○ The others can only proposed \( \text{no-op} \)
  ○ Coordinator of each instance is known by all servers
Deriving Mencius: Simple consensus

- The coordinator proposes \( v \) or \( \text{no-op} \) to all other servers

- The other servers receive proposal from coordinator:
  - \( v \): sends ACCEPT to the coordinator
    - After the coordinator receives ACCEPT from majority, it broadcasts LEARN to all servers
  - \( \text{No-op} \): learn no-op immediately
    - Because only coordinator can propose value other than no-op
Deriving Mencius: Coordinated Paxos

● Actions:
  ○ Suggest:
    ■ Coordinator broadcasts PROPOSE message with requested $v$
  ○ Skip:
    ■ Coordinator broadcasts PROPOSE message with \textit{no-op}
    ● Recipients can learn \textit{no-op} immediately!
  ○ Revoke:
    ■ The coordinator is suspected to be failed
    ■ New leader try to finish the simple-consensus instance by start running a traditional Paxos
Figure 2: The message flow of suggest, skip and revoke in Coordinated Paxos.
Deriving Mencius: Protocol

- **Safety:**
  - Guaranteed by Coordinated Paxos

- **Liveness:**
  - **Rule 1:**
    - Each server $p$ maintains its next simple consensus sequence number $l_p$
    - After receiving request from client, $p$ suggests the request on $l_p$ and update it
Deriving Mencius: Protocol

● Liveness: rule 2:
  ○ Server $p$ receives SUGGEST message with $i >$ current index ($I_p$)
  ○ Before $p$ accepts message, $p$ updates $I_p$ and performs SKIPS for each of the slots in range $[I_p, I'_p)$ that $p$ coordinates.
  ○ $I'_p = \min\{k : p$ coordinates instance $k \land k > i\}$
    ■ Similar to vector clocks!

![Diagram showing the process of updating $I_p$ and performing SKIPS]
Deriving Mencius: Protocol

- Liveness:
  - Rule 3:
    - Server $p$ suspects that server $q$ has failed
    - $C_q$: the smallest instance that is coordinated by $q$ and not learned by $p$
    - $p$ revokes all instances in the range $[C_q, I_p]$ that $q$ coordinates.
  - Rule 4:
    - Server $p$ suggests a value $v \neq no-op$ to instance $i$, and $p$ learns that no-op is chosen
    - $p$ suggests $v$ again
Deriving Mencius: Optimizations

- **Optimization 1:**
  - q sends SUGGEST to p with $i > I_p$
  - p does not send a separate SKIP message to q
  - p uses the ACCEPT message that replies to the SUGGEST to promise that it will not suggest any value to instances smaller than $i$ in the future.

- **Optimization 2:**
  - p does not send a SKIP message to other server r immediately
  - p waits for a future SUGGEST message from p to r to promised that it will not suggest any value to instances smaller than $i$.
  - **PROBLEM:** can defer the propagation of SKIP messages from p to r forever
    - Solution: Accelerator 1
Deriving Mencius: Optimizations

- **Accelerator 1:**
  - A server $p$ propagates SKIP messages to $r$ if either:
    - the total number of outstanding SKIP messages to $r$ is larger than some constant $\alpha$
    - the messages have been deferred for more than some time $\tau$
  - Avoids deferring the SKIP messages forever

- **Optimization 3:**
  - Server $p$ suspects that server $q$ has failed
  - $C_q$: the smallest instance that is coordinated by $q$ and not learned by $p$
  - For some constant $\beta$, $p$ revokes $q$ for all instances in the range $[C_q, I_p + 2\beta]$ that $q$ coordinates if $C_q < I_p + \beta$. 
Deriving Mencius: Recovery

- Temporary broken TCP connection
  - Application-layer sequence number

- Short term failure
  - Stable storage

- Long term failure
  - Checkpoints / state transfer (application specific)
Commit delays

Figure 3: Delayed commit.
Commit delays

- Solution: Out-of-order commits
  - Only works if operations are commutable
  - Implemented by tracking dependencies between requests
  - Experimental results:
    - Slight reduction of latency
    - Particularly effective under low client load
    - Negative effect on throughput when system is CPU-bound
Choosing Parameters

- **α**
  - Limits number of skipped messages for Accelerator 1
  - When $\tau$ is large enough, can reduce overhead for SKIP messages
  - When $\alpha = 20$, SKIP message cost reduced by 95%

- **$\tau$**
  - Amount of time for Accelerator 1
  - Large values allow cost of SKIP messages to be amortized
  - Larger values $\Rightarrow$ higher delay

- **$\beta$**
  - Used in Optimization 3
  - Helps define upper bound for interval of revoked instances
  - Practical lower bounds can be calculated w/experimental data
Evaluation

- C++ implementations of Mencius & Paxos using TCP for transport protocol
- $\alpha = 20$ messages, $\tau = 50$ ms, $\beta = 100,000$ instances.
Conclusion

- Solves bottleneck issues for WANs by letting every server function as a leader
- Achieves high throughput under high client load and low latency under low client load
- Affected more by failures than regular Paxos
  - Paxos suffers performance hits when leader dies; Mencius suffers this when any server dies
Thank you!