Lecture 22: Replications & Consensus

Zhong Shao
Dept. of Computer Science
Yale University

Acknowledgement: some slides are taken from previous lectures by Dr. Ennan Zhai

Lecture Roadmap

• Consistency Issues
• Consistency Models
• Two-Phase Commit
• Consensus
• Case Study: Paxos

Replication Technique

• Distributed systems replicate data across multiple servers

Replication Technique

• Distributed systems replicate data across multiple servers
  - Replication provides fault-tolerance if servers fail

Server1  Server2  Server3
Replica  Replica  Replica

Server1  Server2  Server3
Replica  Replica  Replica

Server1
X

Server1  Server2  Server3
Replica  Replica  Replica

Server1
X

Server1  Server2  Server3
Replica  Replica  Replica
Replication Technique

- Distributed systems replicate data across multiple servers
  - Replication provides fault-tolerance if servers fail
  - Allowing clients to access different servers potentially increasing scalability (max throughput)

What is the problem?

Consistency Problem

- Server1
- Server2
- Server3

Client (Beijing)

Client (DC)

W(X,1)
Consistency Problem

Client (Beijing)

W(X, 1)

Server1

X=1

Server2

X=1

Server3

X=1

R(X) = 1 or 0?

Client (DC)

Consistency Problem

W(X, 1)

Client (Beijing)

X=1

Server1

Client (DC)

X=1

Server2

R(X) = 1

Server3

R(X) = 0

Consistency Problem

Lecture Roadmap

• Consistency Issues
• Consistency Models
  • Two-Phase Commit
  • Consensus
  • Case Study: Paxos
Consistency Models

- A consistency model specifies a contract between programmer and system, wherein the system guarantees that if the programmer follows the rules, data will be consistent.

- A consistency model basically refers to the degree of consistency that should be maintained for the shared data.

- If a system supports the stronger consistency model, then the weaker consistency model is automatically supported.

- But stronger consistency models sacrifice more availability and fault tolerance.
Consistency Models

• Strict consistency
• Strong consistency (Linearizability)
• Sequential consistency
• Causal consistency
• Eventual consistency

These models describe when and how different nodes in a distributed system view the order of operations.

Weaker Consistency Models

Why we have so many consistency models?

They are used for different application scenarios that balance the trade-off between consistency/availability/fault-tolerance.

Strict Consistency

• Strongest consistency model we will consider
  - Any read on a data item X returns value corresponding to result of the most recent write on X

• Need an absolute global time
  - "Most recent" needs to be unambiguous
  - Corresponds to when operation was issued
  - Impossible to implement in real-world (network delays)
Strict Consistency

- Strongest consistency model we will consider
  - Any read on a data item X returns value corresponding to result of the most recent write on X
- Need an absolute global time
  - "Most recent" needs to be unambiguous
  - Corresponds to when operation was issued
  - Impossible to implement in real-world (network delays)

Consistency Models

- Strict consistency
- Strong consistency (Linearizability)
- Sequential consistency
- Causal consistency
- Eventual consistency

These models describe when and how different nodes in a distributed system view the order of operations

Strong Consistency

- Provide behavior of a single copy of object:
  - Read should return the most recent write
  - Subsequent reads should return same value, until next write

Strong Consistency

- Provide behavior of a single copy of object:
  - Read should return the most recent write
  - Subsequent reads should return same value, until next write

- Telephone intuition:
  - 1. Alice updates Facebook post
  - 2. Alice calls Bob on phone: “Check my Facebook post!”
  - 3. Bob reads Alice’s wall, sees her post
Strong Consistency?

Phone call: Ensures happens-before relationship, even though “out-of-band” communication

Cool idea: Delay responding to writes/ops until committed
Strong Consistency? This is buggy!

- Isn't sufficient to return value of server3: It does not know precisely when op is “globally” committed
- Instead: Need to actually order read operation

Strong Consistency!!!

- Order all operations via (1) leader and (2) agreement

Strong Consistency? This is buggy!

- Isn’t sufficient to return value of server3: It does not know precisely when op is “globally” committed
- Instead: Need to actually order read operation

Strong Consistency = Linearizability

- Linearizability:
  - All servers execute all ops in some identical sequential order
  - Global ordering preserves each client’s own local ordering
  - Global ordering preserves real-time guarantee
    - All operations receive global time-stamp via a sync’d clock
    - If TS(x)<TS(y), then OP(x) precedes OP(y) in the sequence
  - Once write completes, all later reads should return value of that write or value of later write
  - Once read returns particular value, all later reads should return that value or value of later write
Intuition: Real-time ordering

- Once write completes, all later reads should return value of that write or value of later write.
- Once read returns particular value, all later reads should return that value or value of later write.

Consistency Models

- Strict consistency
- Strong consistency (Linearizability)
- Sequential consistency
- Causal consistency
- Eventual consistency

These models describe when and how different nodes in a distributed system view the order of operations.

Weaker: Sequential Consistency

Sequential = linearizability - real-time ordering
Weaker: Sequential Consistency
Sequential = linearizability - real-time ordering

- Linearizability:
  - All servers execute all ops in some identical sequential order
  - Global ordering preserves each client's own local ordering
  - Global ordering preserves real-time guarantee
    - All operations receive global time-stamp via a sync'd clock
    - If TS(x)<TS(y), then OP(x) precedes OP(y) in the sequence

Sequential Consistency
- Sequential consistency:
  All (read/write) operations on data store were executed in some sequential order, and the operations of each individual process appear in this sequence

- With concurrent ops, "reordering" of ops acceptable, but all servers must see same order:
  - linearizability cares about time but sequential consistency cares about program order
Implementing Sequential Consistency

- Nodes use vector clocks to determine if two events had distinct happens-before relationship:
  - If timestamp(a) < timestamp(b) => a → b
- If ops are concurrent (i,j, a[i]<b[i] and a[j]>b[j]):
  - Hosts can order ops a, b arbitrarily but consistently

Building Block: Vector Clock

- Initially all clocks are zero
- Each time a process experiences an internal event, it increments its own logical clock in the vector by one
- Each time for a process to send a message, it increments its own clock and then sends a copy of its own vector
- Each time a process receives a message, it increments its own logical clock by one and updates each element in its vector by max(own, received)
**Building Block: Vector Clock**

- **A**: Vector Clock
  - A:0
  - A:1
  - A:2
  - A:3
  - A:4
- **B**: Vector Clock
  - B:0
  - B:1
  - B:2
  - B:3
  - B:5
- **C**: Vector Clock
  - C:0
  - C:1

**Implementing Sequential Consistency**

- Nodes use vector clocks to determine if two events had distinct happens-before relationship:
  - If \( \text{timestamp}(a) < \text{timestamp}(b) \) => \( a \rightarrow b \)
- If ops are concurrent \((i, j, a[i]<b[i] \text{ and } a[j]>b[j])\):
  - Hosts can order ops \( a, b \) arbitrarily but consistently

**Example**

- Host1: OP 1, 2, 3, 4
- Host2: OP 1, 2, 3, 4
- ✔ OP1, OP2, OP3, OP4

**Example**

- Host1: OP 1, 2, 3, 4
- Host2: OP 1, 2, 3, 4
- ✔ OP1, OP2, OP3, OP4
Implementing Sequential Consistency

Host1: OP 1, 2, 3, 4  Host1: OP 1, 3, 2, 4  Host1: OP 1, 2, 3, 4
Host2: OP 1, 2, 3, 4  Host2: OP 1, 3, 2, 4  Host2: OP 1, 3, 2, 4

✔  ✔  ❌

OP1  OP2  OP3  OP4

Sequential Consistency

Server 1  W(x,a)
Server 2  W(x,b)
Server 3  b=R(x)  a=R(x)
Server 4  b=R(x)  a=R(x)

• Is this valid sequential consistency?
  - It is, because Server 3 and 4 agree on order of ops
Sequential Consistency

Server 1 \(W(x,a)\)
Server 2 \(W(x,b)\)
Server 3 \(b=R(x)\), \(a=R(x)\)
Server 4 \(a=R(x)\), \(b=R(x)\)

- Is this valid sequential consistency?
  - No, because Server 3 and 4 do not agree on order of ops.
  - In practice, does not matter when events took place on different machine, as long as server agree on order

Causal consistency

Sequential Consistency

Server 1 \(W(x,a)\)
Server 2 \(W(x,b)\)
Server 3 \(b=R(x)\), \(a=R(x)\)
Server 4 \(a=R(x)\), \(b=R(x)\)

- Is this valid sequential consistency?
  - No, because Server 3 and 4 do not agree on order of ops.
  - In practice, does not matter when events took place on different machine, as long as server agree on order

Sequential Consistency

Server 1 \(W(x,a)\)
Server 2 \(W(x,b)\)
Server 3 \(b=R(x)\), \(a=R(x)\)
Server 4 \(a=R(x)\), \(b=R(x)\)

A valid sequential consistency

Sequential Consistency

Server 1 \(W(x,a)W(x,b)\)
Server 2 \(W(x,b)\)
Server 3 \(b=R(x)\), \(a=R(x)\)
Server 4 \(a=R(x)\), \(b=R(x)\)

A valid sequential consistency or not?
**Sequential Consistency**

- Server 1: \( W(x,a)W(x,b) \)
- Server 2: \( b=R(x) \) \( a=R(x) \)
- Server 3: \( b=R(x) \) \( a=R(x) \)
- Server 4: \( b=R(x) \) \( a=R(x) \)

**A valid sequential consistency or not?**
- No, because it does not preserve local ordering

---

**Consistency Models**

- Strict consistency
- Strong consistency (Linearizability)
- Sequential consistency
- Causal consistency
- Eventual consistency

**Weak consistency model**

These models describe when and how different nodes in a distributed system view the order of operations.

---

**Causal Consistency**

- Causal consistency:
  - Causal consistency is one of weak consistency models
  - Causally related writes must be seen by all processes in the same order
  - Concurrent writes may be seen in different orders on different machines

---

**Causal Consistency**

\[ W(x,a) \]
\[ a=R(x) \]
\[ b=R(x) \]
\[ a=R(x) \]
\[ W(x,b) \]
\[ a=R(x) \]
\[ b=R(x) \]
\[ a=R(x) \]

Not valid

Causally related writes must be seen by all processes in the same order
**Causal Consistency**

- Server 1: \( W(x,a) \) → \( W(x,b) \)
- Server 2: \( b=R(x) \) → \( a=R(x) \)
- Server 3: \( a=R(x) \) → \( b=R(x) \)
- Server 4: \( a=R(x) \) → \( b=R(x) \)

Valid

**Eventual Consistency**

- Eventual consistency:
  - Achieve high availability
  - If no new updates are made to a given data item, eventually all accesses to the data will return the last updated value.
- Eventual consistency is commonly used:
  - Git repo, iPhone sync
  - Dropbox and Amazon Dynamo

**Consistency Models**

- **Strict consistency**
- **Strong consistency (Linearizability)**
- **Sequential consistency**
- **Causal consistency**
- **Eventual consistency**

**Weak consistency model**

These models describe when and how different nodes in a distributed system view the order of operations.

- **Strict consistency**
- **Strong consistency (Linearizability)**
- **Sequential consistency**
- **Causal consistency**
- **Eventual consistency**

**Weak consistency model**

These models describe when and how different nodes in a distributed system view the order of operations.
Lecture Roadmap

• Consistency Issues
• Consistency Models
• Two-Phase Commit
• Consensus
• Case Study: Paxos

Two-Phase Commit

• Goal: Reliably agree to commit or abort a collection of sub-transactions

• All the operations happens at single master node
  - Concurrent machines
  - Failure and recovery of machines

  Achieve strong consistency!

Intuitive Example

• You want to organize outing with 3 friends at 6pm Tue
  - Go out only if all friends can make it

• What do you do?
  - Call each of them and ask if can do 6pm Tue (voting phase)
  - If all can do Tue, call each friend back to ACK (commit)
  - If one cannot do Tue, call others to cancel (abort)

Intuitive Example

• You want to organize outing with 3 friends at 6pm Tue
  - Go out only if all friends can make it

• What do you do?
  - Call each of them and ask if can do 6pm Tue (voting phase)
  - If all can do Tue, call each friend back to ACK (commit)
  - If one cannot do Tue, call others to cancel (abort)

  This is exactly how two-phase commit works
Two-Phase Commit Protocol
• Phase 1: Voting phase
  - Get commit agreement from every participant

  Coordinator
  → Participant
  → Participant
  → Participant

  commit?
  commit?
  commit?

  Coordinator
  → Participant
  → Participant
  → Participant

  yes
  yes
  yes

Two-Phase Commit Protocol
• Phase 2: Commit phase
  - Send the results of the vote to every participant
  - A single "no" response means that we will have to abort

  Coordinator
  → Participant
  → Participant
  → Participant

  yes
  yes
  yes

  Coordinator
  → Participant
  → Participant
  → Participant

  commit
  commit
  commit

  Coordinator
  → Participant
  → Participant
  → Participant

  commit
  commit
  commit
Two-Phase Commit Protocol

- Phase 2: Commit phase
  - Get “committed” acknowledgements from every participant

Two-Phase Commit Protocol

- Two-phase commit assumes a fail-recover model
  - Any failed system will eventually recover

  - A recovered system cannot change its mind
    - If a node agreed to commit and then crashed, it must be willing and able to commit upon recovery

  - If the leader fails?
    - Lose availability: system not longer “live”

Lecture Roadmap

- Consistency Issues
- Consistency Models
- Two-Phase Commit
- Consensus
- Case Study: Paxos

Consensus / Agreement Problem

- Definition:
  - A general agreement about something
  - An idea or opinion that is shared by all the people in a group

- Given a set of processors, each with an initial value:
  - **Termination**: All non-faulty processes eventually decide on a value
  - **Agreement**: All processes that decide do so on the same value
  - **Validity**: The value that has been decided must have been proposed by some process
Consensus / Agreement Problem

- Goal: N processes want to agree on a value

- Correctness (safety):
  - All N nodes agree on the same value
  - The agreed value has been proposed by some node

- Fault-tolerance:
  - If <= F faults in a window, consensus reached eventually
  - Liveness not guaranteed: If > F faults, no consensus

  Given goal of F, what is N? Depends on fault model
  ("Crash fault" need 2F+1; Byzantine fault needs 3F+1)

Lecture Roadmap

- Consistency Issues
- Consistency Models
- Two-Phase Commit
- Consensus
- Case Study: Paxos
**Paxos**

- **Safety:**
  - Only a single value is chosen
  - Only a proposed value can be chosen
  - Only chosen values are learned by processes

- **Liveness:**
  - Some proposed value eventually chosen if fewer than half of processes fail
  - If value is chosen, a process eventually learns it

**Paxos + Two-Phase Commit**

- **Use Paxos for view-change**
  - If anybody notices current master unavailable, or one or more replicas unavailable
  - Propose view change Paxos to establish new group: Value agreed upon = \(<2PC\ Master, \{2PC\ Replicas\}>\).

- **Use two-phase commit for actual data**
  - Writes go to master for two-phase commit
  - Reads go to acceptors and/or master