Distributed Systems

Consistency Models

WEEK 10: Data Consistency, Strong Consistency, Eventual Consistency, Client-Centric Consistency, Consistency Protocols.

Online Lecture Series - 10

Felix Edesa

Addis Ababa Science and Technology University

AASTU

Course Outline: Consistency Models (Lecture 10)

Topics We Will Cover

- **Data Consistency** — Introduction, importance in distributed systems
- **Strong Consistency** — Definition, sequential consistency, linearizability
- **Eventual Consistency** — Definition, trade-offs, examples in distributed storage
- **Client-Centric Consistency** — Monotonic Reads/Writes, Read-Your-Writes, Writes-Follow-Reads
- **Consistency Protocols** — Overview of protocols supporting different models, including primary-based and replicated-write protocols
- **Examples and Applications** — Practical scenarios demonstrating each consistency model

Data Consistency: Introduction

Definition

- Data consistency ensures all nodes in a distributed system have a coherent view of shared data.
- Guarantees that operations on data produce predictable and reliable results.
- Critical for correctness in distributed applications.

Examples

- Online banking: all branches see updated balances after transactions.
- Inventory system: real-time stock updates prevent overselling.

AASTU

Importance of Data Consistency

Why it matters

- Ensures reliability in distributed operations.
- Prevents conflicts and anomalies in shared data.
- Supports trust in multi-user and multi-location systems.
- Enables predictable behavior for critical applications.

Use Cases

- Collaborative editing: multiple users working on same document.
- Financial systems: accurate and consistent transaction records.

AASTU

Consistency Challenges in Distributed Systems

Challenges

- Network latency can cause nodes to have outdated information.
- Concurrent updates may lead to conflicts.
- Partial failures make synchronization difficult.
- Maintaining global state in large-scale systems is complex.

Example

- Two users edit the same record simultaneously: potential data conflict.
- Distributed database may temporarily show inconsistent values.

AASTU

Consistency vs Availability

Trade-Offs

- CAP theorem: consistency, availability, partition-tolerance — pick 2 of 3.
- Strict consistency may reduce availability during network partitions.
- Eventual consistency improves availability at the cost of temporary inconsistency.

Example

- Social media feed: may show slightly outdated posts to ensure availability.
- Banking system: prefers strict consistency to avoid overdraft errors.

AASTU

Case Study: Banking System

Scenario

- Multiple branches of a bank share the same database.
- Transfers must reflect in all locations immediately.
- Consistency mechanisms ensure no branch sees stale balances.

Illustration

- Customer transfers $500 from branch A to branch B.
- All nodes must update balance synchronously to prevent overdraft.

AASTU

Distributed E-Commerce Inventory

Challenge

- High-concurrency updates to product stock during sales.
- Overselling occurs if updates are inconsistent.
- Requires synchronization across multiple warehouses or servers.

Example

- Flash sale: 100 units of a product available across 3 warehouses.
- Consistency ensures no more than 100 units are sold in total.

AASTU

Collaborative Editing Applications

Overview

- Multiple users edit the same document concurrently.
- Consistency ensures changes are visible in correct order.
- Conflict resolution strategies maintain a coherent document state.

Example

- Google Docs: changes from User A and User B merged in real time.
- OT (Operational Transformation) or CRDTs ensure consistency.

AASTU

Sensor Networks and IoT

Importance

- Sensors often update shared environmental data.
- Consistency ensures monitoring systems receive coherent readings.
- Crucial for automated control systems and alerts.

Example

- Smart building: temperature sensors report consistent readings for HVAC control.
- IoT devices in agriculture: soil moisture levels must be synchronized.

AASTU

Event Logs and Audit Trails

Consistency Importance

- Distributed logs must be consistent to reconstruct events correctly.
- Auditing requires reliable, ordered data from all nodes.
- Ensures accountability and debugging accuracy.

Example

- Microservices: each service logs transactions to a distributed log.
- Consistency allows detecting the exact sequence of operations.

AASTU

Lessons Learned & Best Practices

Key Takeaways

- Design distributed systems with consistency requirements in mind.
- Identify critical data requiring strong consistency.
- Use eventual consistency for less critical data to improve availability.
- Apply conflict resolution and monitoring for concurrent updates.

Best Practices

- Balance consistency and availability based on application needs.
- Test systems under concurrent updates and network partitions.
- Use strong consistency only where correctness is essential.

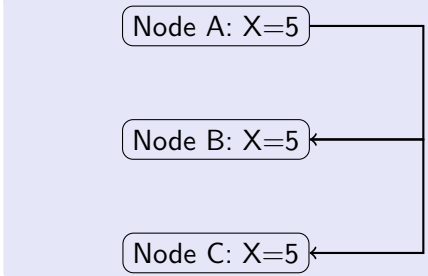
AASTU

Strong Consistency: Definition

Definition

- All nodes see the same data at the same time.
- Operations appear atomic across the system.
- Ensures predictable behavior for concurrent clients.
- Example: Bank account balance updates must be reflected instantly.

Mini Diagram



Sequential Consistency: Concept

Principles

- Operations appear in some sequential order.
- All processes agree on the order of operations.
- Local order of operations is preserved per process.
- Example: Two clients write to X; everyone sees X updated in same sequence.

Diagram Example

Sequential Consistency: Problem and Solution

Problem Example

- P1 writes X=5, P2 writes X=10 concurrently.
- Client C1 reads X=5, C2 reads X=10.
- Without order agreement, inconsistencies appear.
- Solution: enforce sequential consistency, agree on operation order.

Mathematical Example

- Operation sequence: W(P1)=5, W(P2)=10.
- Valid sequential views:

C1,C2 see X=5 → X=10
C1,C2 see X=10 → X=5

- Guarantees linear sequence of operations.

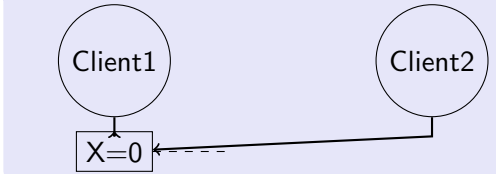
AASTU

Linearizability: Definition

Key Points

- Operations appear atomic between invocation and response.
- Stronger than sequential consistency.
- Preserves real-time order globally.
- Example: Shared counter increment; increments appear instantaneous.

Diagram



Linearizability: Problem and Solution

Problem Example

- Shared variable X updated by Client1 and Client2.
- Without linearizability, reads may see inconsistent values.
- E.g., X=5 read before X=10 committed.
- Solution: enforce linearizability, update appears instantaneous.

Practical Use Case

- Bank transfer: deduct from A, credit to B. Must appear atomic.
- Real-time monitoring: sensor values globally consistent.

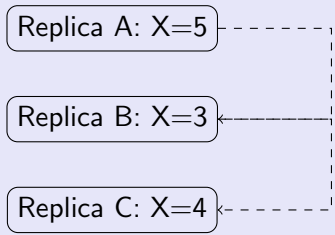
AASTU

Eventual Consistency: Definition

Definition

- Updates to a replicated object propagate asynchronously.
- All replicas converge eventually if no new updates occur.
- Strong consistency not guaranteed at all times.
- Example: Amazon DynamoDB, Cassandra.

Illustration



Eventual Consistency: Trade-offs

Key Trade-offs

- High availability even during network partitions.
- Lower latency for writes.
- Temporary inconsistencies possible across replicas.
- Applications must tolerate stale reads.

Example Scenario

- User updates profile from Node A; Node B sees old value temporarily.
- Eventually, all nodes converge to latest profile.

AASTU

Eventual Consistency: Mathematical Example

Propagation Calculation

- Suppose three replicas: $X_A = 5, X_B = 3, X_C = 4$
- Updates propagate every 1s.
- Convergence rule: $X_i = \max(X_i, X_j)$ for neighbors
- After 2 steps, $X_A = X_B = X_C = 5$

Interpretation

- Convergence is guaranteed.
- Temporary inconsistencies acceptable.
- Example: counters in distributed databases.

AASTU

Eventual Consistency: Problem and Solution

Problem

- Two users increment a distributed counter concurrently.
- Replica A sees $+1$, Replica B sees $+2$.
- Without resolution, global counter is inconsistent.
- Solution: merge updates using last-write-wins or CRDTs.

Practical Example

- Amazon DynamoDB counters, shopping cart updates.
- All replicas eventually reflect total count correctly.

AASTU

Client-Centric Consistency: Introduction

Overview

- Focuses on **consistency guarantees from a single client's perspective**.
- Useful in distributed systems where global strong consistency is expensive.
- Ensures operations appear consistent **for the client** even if replicas differ.
- Types include:
  - Monotonic Reads / Writes
  - Read-Your-Writes
  - Writes-Follow-Reads

Practical Examples

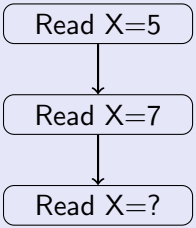
- Social media: your own posts are immediately visible to you.
- Cloud document editing: your changes appear in order.
- E-commerce: shopping cart updates reflect immediately to the same client.

AASTU

Monotonic Reads

Definition Diagram

- Subsequent reads never return an older value.
- Maintains **client-side read ordering**.



Example

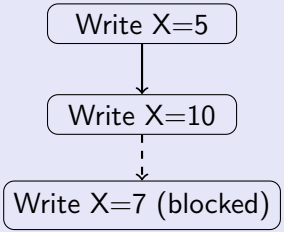
- Reads bank balance X=500, then X=550.
- Cannot see X=500 again (monotonic).
- Ensures **transaction consistency**.
- Prevents anomalies in sequential reads.

AASTU

Monotonic Writes

Definition Diagram

- Client writes are applied **in order**.
- Prevents later writes from appearing before earlier ones.



Example Application

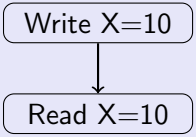
- Collaborative doc: update para 1, then para 2.
- Prevents out-of-order overwrites.
- Ensures **consistent write history** for the client.
- Sequence numbers or timestamps enforce order mathematically.

AASTU

Read-Your-Writes

Definition Diagram

- Client always reads its **latest writes**.
- Prevents seeing stale data after performing a write.



Example Application

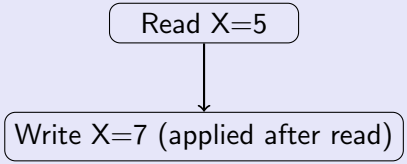
- Client updates shopping cart: add item A.
- Next read shows updated cart including A.
- Ensures **session consistency** in cloud apps.
- Prevents confusion in e-commerce workflows.

AASTU

Writes-Follow-Reads

Definition Diagram

- Ensures writes after a read are applied **after the latest read**.
- Prevents writes based on stale data.



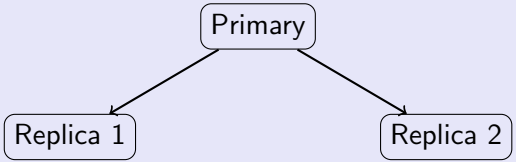
Example

- Client reads bank balance X=500.
- Deposits 50 $\rightarrow$ X=550.
- Ensures write reflects latest read.
- Replicas converge correctly.

Consistency Protocols: Overview

Overview

- Protocols ensure consistency across replicas.
- Primary-based, Replicated-write
- Trade-offs: consistency, availability, latency.



Key Points

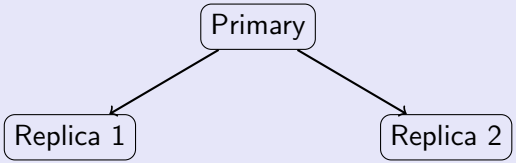
- Primary-based = simple consistency.
- Replicated-write = higher availability.
- Choice depends on application requirements.

AASTU

Primary-Based Protocols

Mechanism and Example

- Primary handles all writes; replicas follow.
- Ensures strong consistency: all replicas see same value.
- Example: Bank account X=100 update.



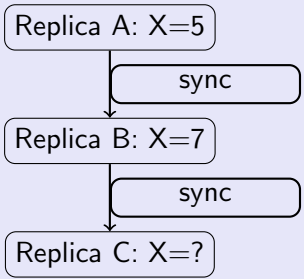
Applications

- Banking systems
- Reservation systems
- Strong consistency required apps

Replicated-Write Protocols

Concept

- Multiple replicas can handle write requests.
- Requires conflict resolution.



Use Cases

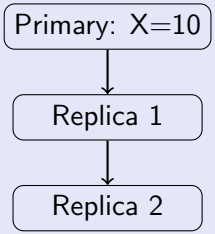
- Used in DynamoDB and Cassandra.
- Ideal for high-availability systems.
- Balances latency and eventual consistency.

AASTU

Primary-Based Protocols

Concept

- Writes go to a single **primary replica**.
- Updates sync to secondary replicas.
- Ensures strong data consistency.



Applications

- Used in leader-based systems (e.g., HDFS, Kafka).
- Simplifies synchronization and conflict handling.
- Trade-off: possible delay if the primary fails.

AASTU

Primary-Based Protocols: Example and Math

Mathematical Example

- Let primary update $X = X + 2$.
- Replicas synchronize every $\Delta t = 2s$.
- If network delay = 1s, replicas lag by 1s behind primary.

$$X_p = 10, \quad X_r(t+1) = X_p(t)$$

Observation

- Predictable delay-based convergence.
- Great for monitoring and financial data.

Synchronous vs Asynchronous Replication

Comparison

- **Synchronous:** Waits for all replicas to confirm update.
- **Asynchronous:** Writes committed locally, propagated later.
- Example: Financial systems → synchronous; Social feeds → async.

Trade-offs

- Sync → High latency, strong consistency.
- Async → Low latency, eventual consistency.

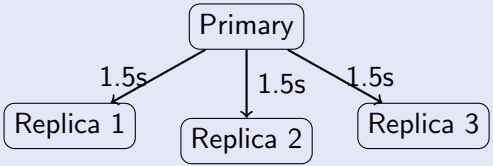
AASTU

Problem: Replication Delay and Consistency

Problem Illustration

Primary updates shared data every 2 seconds. Each of 3 replicas has a propagation delay of 1.5s. Determine how long it takes for all replicas to reach consistency.

$$T_c = n \times d = 3 \times 1.5 = 4.5\text{s}$$



Discussion & Solution

- All replicas consistent after **4.5s**.
- Real systems use parallel updates $\rightarrow$ faster sync.
- High delays cause **temporary inconsistency**.
- Applied in *database replication* and *cloud backup systems*.

AASTU

Quorum-Based Protocols

Concept & Example

- System ensures consistency using read/write quorums:

$$R + W > N$$

- Overlap guarantees at least one shared node for coordination.
- Example: $N = 5$, choose $W = 3$, $R = 3 \rightarrow$ overlap $= 1$.

Write quorum $W = 3$, Read quorum $R = 3$

Use Cases

- Applied in **DynamoDB**, **Cassandra**, and **Riak**.
- Provides a trade-off between **availability** and **consistency**.
- Larger $W \rightarrow$ stronger consistency; smaller $W \rightarrow$ faster writes.

Mathematical Example — Quorum Consistency

Computation Example

- Total replicas $N = 7$
- Write quorum $W = 4$, Read quorum $R = 4$

$R + W = 8 > 7 \Rightarrow$ Strong Consistency

If $W = 2, R = 2 \Rightarrow R+W = 4 < 7 \Rightarrow$ Event- Cons

Quorum nodes for write ($W = 4$)

Insights

- Strong consistency when $R + W > N$.
- Smaller quorums improve latency but risk stale reads.
- Used to tune performance in systems like DynamoDB.

AASTU

Real-World Applications of Consistency Protocols

Case Study

- **Google Spanner:** TrueTime API $\rightarrow$ global linearizability.
- **Cassandra:** Tunable consistency via quorum settings.
- **Etcd/ZooKeeper:** Primary-based, consensus through Raft.

Takeaways

- Choice of protocol affects scalability and reliability.
- Trade-offs between latency, fault-tolerance, and correctness.

AASTU

Examples and Applications — Overview

Concept Summary

- Consistency models define how data updates appear to clients.
- Choice of model depends on performance, latency, and reliability trade-offs.
- Systems may use hybrid approaches for different operations.

Applications Overview

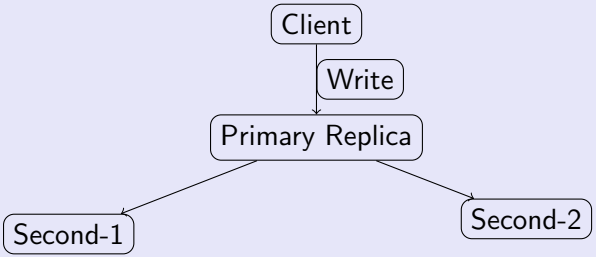
- Banking $\rightarrow$ Strong consistency.
- Social media $\rightarrow$ Eventual consistency.
- E-commerce $\rightarrow$ Client-centric consistency.

AASTU

Strong Consistency — Banking Example

Scenario

- Account balance updated across all replicas **before** confirmation.
- Prevents double spending and stale data reads.



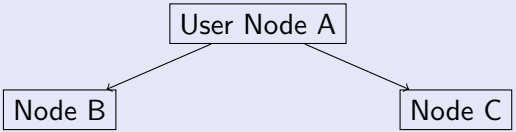
Application

- Used in ATM transactions and fund transfers.
- Requires strict synchronization.
- Mathematical model:
  $T_{commit} = T_{write} + T_{replicate}$

Eventual Consistency — Social Media Feeds

Scenario

- User posts a photo on Node A.
- Friends on Node B or C may see it after few seconds.
- Updates propagate asynchronously.



Application

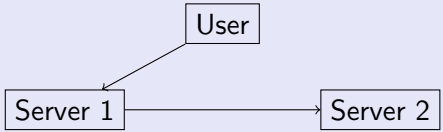
- Used in Facebook, Instagram, and Twitter.
- Mathematical model:

$$T_c = \max(T_{propagate}) < T_{user\ \ tolerance}$$

Client-Centric Consistency — E-commerce Cart

Scenario

- User adds items from multiple regions.
- Each server ensures user sees their latest state.
- Provides session-level consistency.



Application

- Amazon and Alibaba use session consistency.
- Prevents cart loss across multiple devices.

AASTU

Questions?

Thank you for your attention!

References

Suggested References

1. M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., Version 4.02, February 2024.
2. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed., Addison-Wesley, 2012.
3. N. Lynch, *Distributed Algorithms*, 2nd ed., Morgan Kaufmann, 1996.
4. R. E. Bryant and D. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 4th ed., Pearson, 2024.

AASTU