Models of Consistency in Distributed Systems

Introduction

In distributed systems, **consistency models** define the rules and guarantees about how and when updates to shared data are propagated and made visible across the system. The choice of a consistency model impacts the system's performance, scalability, and correctness. Consistency models are particularly relevant in distributed databases, shared memory systems, and cloud computing.

1. Types of Consistency Models

Consistency models can be broadly categorized into **strict consistency** and **relaxed consistency**.

1.1 Strict Consistency

- **Definition**: A system is strictly consistent if a read operation always returns the result of the most recent write operation, regardless of which process performs the read.
- **Mathematical Representation**:

  $$R(x) \to W(x)$$

  Where:

  - $R(x)$: Read operation on variable x.
  - $W(x)$: Write operation on variable x.
  - $R(x) \to W(x)$: The read operation $R(x)$ returns the result of the most recent write $W(x)$.

1.2 Sequential Consistency

- **Definition**: Operations from all processes appear to be executed in some sequential order, and each process's operations are executed in program order.
- **Mathematical Representation**:

  $$\forall P : \exists \text{order } O, \text{ such that } O = P_1 + P_2 + \cdots + P_n$$

  Where:

  - P: Process operations.
  - O: A global sequential order of operations.

1.3 Causal Consistency

- **Definition**: If one operation causally affects another, all processes must observe the operations in the same causal order.
- **Mathematical Representation**:

$$W(x)_i \to W(x)_j \implies R(x)_k \text{ observes } W(x)_i \text{ before } W(x)_j$$

  Where:

  - $W(x)_i \to W(x)_j$: Write $W(x)_i$ causally precedes $W(x)_j$.
  - $R(x)_k$: Read operation observes writes in causal order.

1.4 Eventual Consistency

- **Definition**: If no new updates are made to a replicated data item, eventually all replicas will converge to the same value.
- **Mathematical Representation**:

$$\forall R(x) : R(x) \to \text{converge to the same value}$$

  Where:

  - $R(x)$: Read operations on x.

2. Relaxed Consistency Models

2.1 Weak Consistency

- **Definition**: Guarantees consistency only at synchronization points explicitly defined by the application.
- **Mathematical Representation**:

$$\text{Consistency guaranteed at } S(x)$$

  Where:

  - $S(x)$: Synchronization points.

2.2 Release Consistency

- **Definition**: Divides synchronization into two types:
  1. **Acquire**: Guarantees that updates made by other processes before the acquire are visible.
  2. **Release**: Ensures that updates made before the release are visible to other processes.
- **Mathematical Representation**:

$$R(x)_{\text{acquire}} \leq W(x)_{\text{release}}$$

2.3 Lazy Consistency

- **Definition**: Updates are propagated lazily to replicas, allowing temporary inconsistencies.
- **Mathematical Representation**:

$$\Delta t \text{ delay in propagation}$$

  Where:

  - Δt: Time delay for updates to be visible across replicas.

3. Formal Definitions Using Happens-Before Relation

The **happens-before relation** ($\rightarrow$) helps define consistency models:

1. **Strict Consistency**:

$$x \rightarrow y \implies x \text{ visible before } y$$

2. **Causal Consistency**:

$$x \rightarrow y \implies \forall P, x \text{ observed before } y$$

3. **Sequential Consistency**:

$$\forall P : \exists \text{ total order consistent with process order}$$

4. Consistency Metrics

4.1 Staleness

Measures the difference between the most recent write and the value observed by a read:

$$\text{Staleness} = T_{\text{read}} - T_{\text{last write}}$$

4.2 Consistency Cost

The communication cost to maintain consistency:

$$C_{\text{consistency}} = N \cdot T_{\text{sync}}$$

Where:

- N: Number of replicas.
- T_{sync}: Synchronization cost per replica.

5. Trade-offs in Consistency

5.1 CAP Theorem

The CAP theorem states that a distributed system can provide at most two out of the following three:

1. **Consistency**: All nodes see the same data at the same time.
2. **Availability**: Every request receives a response, regardless of system state.
3. **Partition Tolerance**: The system continues to operate despite network partitions.

5.2 PACELC Model

The PACELC model extends the CAP theorem, addressing the trade-off between:

- **Latency (L)** and **Consistency (C)** during normal operations.
- **Partition Tolerance (P)** and **Consistency (C)** during network partitions.

6. Applications of Consistency Models

1. **Strict Consistency**:
    - Real-time systems, financial transactions.
2. **Sequential Consistency**:
    - Shared memory in multiprocessor systems.
3. **Eventual Consistency**:
    - Distributed databases (e.g., DynamoDB, Cassandra).

Conclusion

Consistency models define how updates and reads behave in distributed systems. While strict consistency provides strong guarantees, relaxed models like eventual consistency improve scalability and performance. Mathematical models help formalize these concepts, enabling system designers to balance consistency, availability, and partition tolerance based on application requirements.