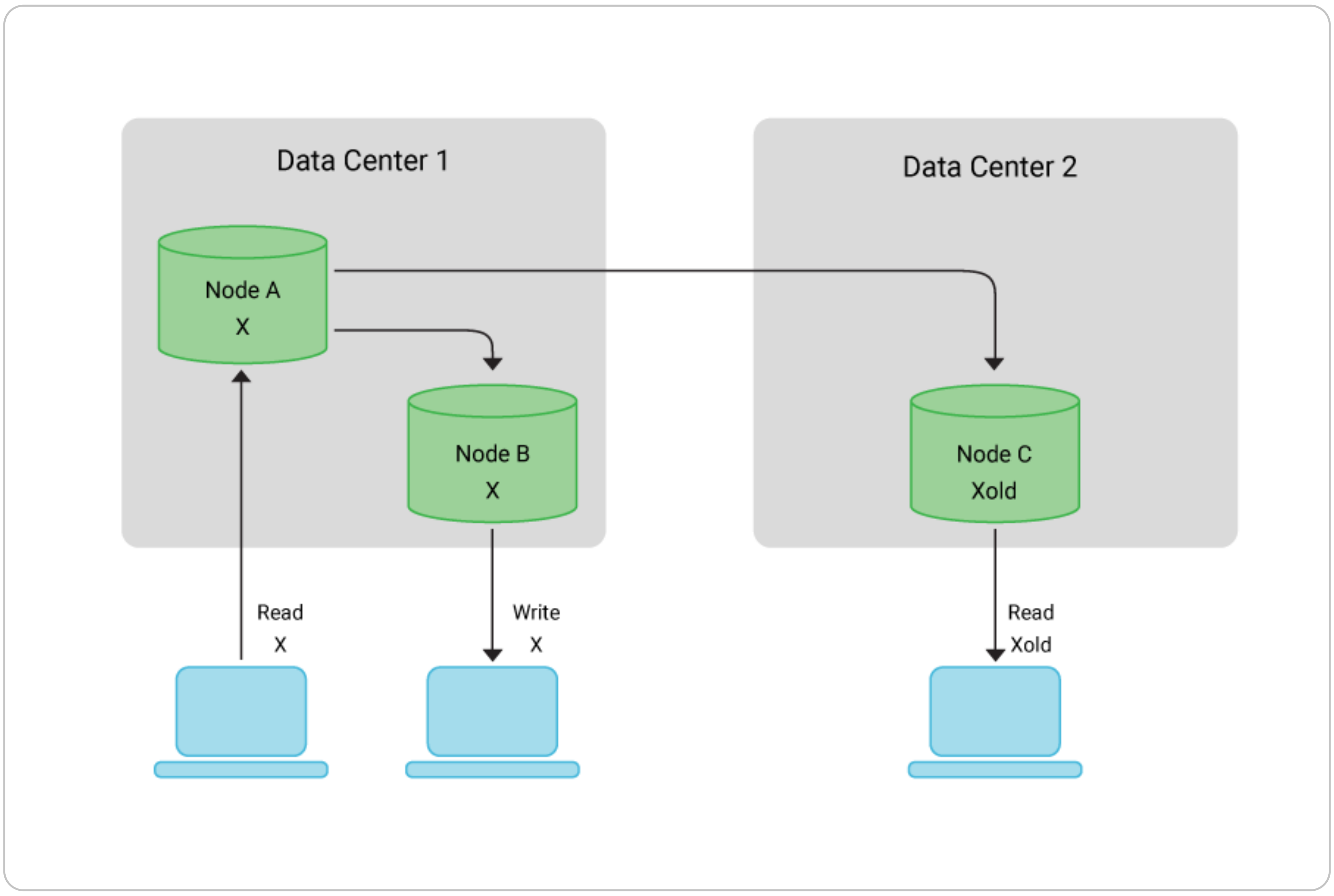
Consistency Model

Consistency Models Definition

In the context of computer science, a consistency model refers to a set of rules or constraints that dictate how shared data should appear to processes in a distributed system. It provides a way to reason about the ordering and visibility of operations (reads and writes) on shared data across different processes or nodes within a network.



Consistency Models FAQs

What are the Dominant Models of Data Consistency?

Distributed systems often replicate data across multiple nodes, sometimes in real-time, to improve fault tolerance and performance. However, this replication introduces challenges in maintaining data consistency. The consistency model defines specific guarantees the system provides for how it synchronizes these replicated copies of data and how different processes can access the copies.

There are several data consistency models, each offering different trade-offs between data consistency and system performance. Some common consistency models include:

Strong consistency. All operations appear to be executed in some total order. Each read operation will return the value of the most recent write to the same data item. This provides the strongest guarantee of consistency, but can potentially impact system performance due to synchronization overhead.

Eventual consistency. The system guarantees that if no new updates are made to a data item, all replicas will converge to the same value eventually. It allows replicas to diverge temporarily and reconcile later, making it more scalable but providing weaker consistency guarantees.

Causal consistency. This model ensures that where operations that are causally related, for example where one operation happens before another, they will be seen in the same order by all processes. However, there may be no total ordering of all operations.

Read-your-writes consistency. This model guarantees that if a process performs a write and then performs a read, it will always see the value it previously wrote.

Monotonic reads/writes consistency. These models guarantee that processes that perform sequences of reads (or writes) will not see older versions of data later in the execution.

In the context of classifying consistency models in distributed systems, there are two different approaches used to define and understand consistency models:

Issue method. The issue method focuses on the sequence or order of operations (reads and writes) performed by processes in the distributed system, how they interact with each other, and how operations are scheduled and executed by processes, leading to various possible outcomes in terms of consistency.

In the issue method, each process performs a sequence of operations, and the consistency model defines rules and constraints on how these operations can be interleaved with operations from other processes. It provides a way to reason about the possible orderings and dependencies between operations and their effects on data consistency. Strong consistency and sequential consistency are both common consistency models defined using the issue method.

View method. The view method focuses on the observations or views that processes have of the system's state in the form of the timing and visibility of data changes from operations performed by one process that affects others in the distributed system.

In the view method, each process executes operations and observes the state of the system at different points in time. The consistency model defines rules and constraints on how the system's state can transition from one valid state to another, ensuring that the views of different processes are consistent with each other. Read-your-writes consistency and monotonic reads consistency are both common consistency models defined using the view method.

Consistency Models in Distributed Systems

Strong and weak consistency models offer different levels of guarantees for the order and visibility of operations on shared data in a distributed system.

Strong consistency is a stringent consistency model that ensures that all operations on shared data appear to take place in some total order. The system behaves as if there is a single copy of the data, and all operations are executed one after another without any concurrent or out-of-order effects.

Strongly consistent systems are characterized by the following properties:

Sequential consistency. The outcome of any execution is the same as if all processes and operations were executed in sequential order, and the operations of each individual process appear in the order specified by its program.
Linearizability. This stronger property than sequential consistency ensures that the outcome of each operation appears to take effect instantaneously at some point between the operation's invocation and response, as if there is a global timeline for all operations.

Strong consistency provides the highest level of data consistency but may come at the cost of increased latency and coordination overhead in a distributed system. Ensuring strong consistency often requires synchronization mechanisms, such as locking or consensus protocols, to enforce a strict ordering of operations.

Weak consistency is a more relaxed consistency model that allows for more flexibility in the ordering and visibility of operations on shared data. In contrast to strong consistency, weak consistency does not guarantee that all operations appear in a total order. Instead, it allows for some degree of concurrency and potential divergence between replicas of the data.

Weakly consistent systems are characterized by the following properties:

Eventual consistency. This guarantees that if no new updates are made to a data item, all replicas of that item will eventually converge to the same value. However, during periods of network partitions or delays, replicas might temporarily diverge.
Causal consistency. A causal consistency model ensures that operations that are causally related (i.e., one operation happens before another) will be seen in the same order by all processes. However, there might be no total ordering of all operations.

Weak consistency models are often favored in distributed systems that prioritize availability and partition tolerance over strong consistency (see the CAP Theorem). They can handle system partitions and allow for better scalability and performance as replicas can operate independently for extended periods.

Examples of of Consistency Models in Distributed Systems

There are many consistency models used in distributed systems today. Here are a few common examples.

Consistency Models of NoSQL Databases

Eventual consistency is a common weak NoSQL consistency model employed by many NoSQL databases. It allows for replicas of data to diverge temporarily but guarantees that, if no new updates are made to a data item, all replicas will eventually converge to the same value. This convergence typically happens over time, as the system automatically propagates updates from one replica to others.

Although NoSQL databases are often associated with eventual consistency, some NoSQL consistency models provide options for strong consistency guarantees as well. In such cases, the database ensures that all replicas of a data item have the same value and operations appear to take place in a strict total order.

DynamoDB consistency models

By default, DynamoDB uses “Eventually Consistent Reads” for read operations. This consistency model allows for low-latency and high-throughput reads but provides a relaxed consistency guarantee. DynamoDB also offers the option of “Strongly Consistent Reads” for applications that require immediate and up-to-date data.

MongoDB consistency models

MongoDB offers strong consistency for read and write operations. When a write operation is acknowledged, the data is immediately visible to all subsequent read operations. MongoDB also supports eventual consistency for read operations.

Cassandra consistency models

Apache Cassandra essentially offers tunable consistency levels ranging from strong to eventual. However, for most use cases, there is a default Cassandra eventual consistency model that works well.

ACID and BASE Consistency Models

The ACID and BASE consistency models are two contrasting approaches to achieving data consistency in distributed systems.

ACID Consistency Model

The ACID consistency model stands for atomicity, consistency, isolation, and durability. It represents a set of properties that guarantee the reliability and integrity of transactions in a database system.
Atomicity. This property ensures that a transaction is treated as a single unit of work. Either all of its operations are successfully completed, or none of them are applied. Should the transaction fail in whole or part, the entire transaction will be rolled back, and the database will be returned to its original state.
Consistency. ACID consistency ensures that a transaction brings the database from one valid state to another. It enforces integrity constraints, business rules, and data validations, ensuring that the database remains in a consistent state throughout the transaction's execution.
Isolation. Isolation ensures that multiple transactions can run concurrently without interfering with each other. Each transaction is isolated from others until it is committed, preventing dirty reads, non-repeatable reads, and other

anomalies.
Durability. Durability ensures that after a transaction is committed, those changes are permanent and survive system crashes or any other subsequent failures. Committed data is stored safely and can be retrieved even in the event of a power outage or hardware failure.

BASE Consistency Model

The BASE consistency model stands for basically available, soft state, and eventually consistent. It represents a more relaxed approach to consistency, prioritizing availability and partition tolerance over strong consistency guarantees.
Basically Available: BASE consistency ensures that the database remains available for both read and write operations, even in the presence of failures or partitions. High availability is a key feature of BASE systems.
Soft state. Soft state means that the data in the system can change over time without strict consistency guarantees. The system allows temporary inconsistencies among replicas but expects the data to converge eventually.
Eventually consistent. BASE systems embrace eventual consistency, so if no new updates are made to data, all replicas will eventually converge to the same value. It allows some degree of data divergence but ensures that the system eventually reaches a consistent state.

ACID vs BASE Comparisons

Consistency Guarantees

- ACID provides strong consistency guarantees, ensuring that the data is always in a consistent state.
- BASE provides eventual consistency, allowing temporary inconsistencies but ensuring convergence to a consistent state over time.

Availability

- ACID systems may experience limited availability during certain transactions or under system failures, as they prioritize consistency.

- BASE systems prioritize availability and strive to remain accessible even during failures or network partitions.

Performance

- ACID systems may incur higher latency and performance overhead, requiring synchronous replication and strict consistency enforcement.
- BASE systems can achieve higher throughput and lower latency due to their asynchronous replication and relaxed consistency.

Use Cases

- ACID is well-suited for applications that require strong data integrity and consistency, such as financial systems or transactional applications.
- BASE is often used in large-scale distributed systems, NoSQL databases, and web applications where high availability and horizontal scalability are more critical than strict consistency.