cloudacademy.com

How to Use Consistency Models for Amazon Web Services

Chandan Patra

7-9 minutes

If you're interested in learning how consistency models on AWS can help you write stable, reliable applications, then this is the article for you. By following a consistency model, your application's memory will remain consistent and the results of any operations on its memory should be predictable. (*Editor's Note*: This is complex material. If you'd like to brush up on your understanding of storage in AWS, check out this course.)

Consistency Models Create Structure and Rules around Memory to Ensure Application Reliability

In very simple terms, **consistency models define rules for the order and visibility of read and updates**.

Distributed systems are large and replicated across many servers, allow concurrent execution of components, are prone to failure, experience transaction delays, and have no global time. Objects in a distributed storage system are replicated to avoid single-point failures and improve both reliability and availability to avoid overload of transactions in a single system and to give faster access to local copies to avoid communication delay.

But all these virtues of a distributed system come at a price as multiple copies of data need to be kept identical. This requirement brought the necessities of a suitable consistency model for different distributed services such as storage, memory, or a NoSQL offering.

Broadly speaking, there are two types of consistency models: **Data-centric** and **client-centric**. Let's take a look at both of them.

Data-Centric Consistency Models

Tanenbaum & Maarten Van Steen, two computer scientists who are experts in this field, define the consistency model **as a contract between the software (processes) and memory implementation (data store)**. This model guarantees that if the software follows certain rules, the memory works correctly. Since, in a system without a global clock, defining the last operation writes is difficult, some restrictions should be applied on the values that can be returned by a read operation.

The following models are the data-centric consistency models according to their strictness in descending order – **the strictest models are listed first**:

Models	**Description**
Strict Consistency	Absolute time ordering of all shared accesses matters
Linearizability Consistency	All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (non-unique) global timestamp
Sequential Consistency	All processes see all shared accesses in the same order. Accesses are not ordered in time
Causal	All processes see causally-related shared

Consistency	accesses in the same order.
FIFO Consistency	All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order
Weak Consistency	Shared data can be counted on to be consistent only after a synchronization is done
Release Consistency	Shared data are made consistent when a critical region is exited
Entry Consistency	Shared data pertaining to a critical region are made consistent when a critical region is entered.

Client-Centric Consistency Models

In a client-centric consistency model, the emphasis is put on how data is seen by the clients. The data can be varying from clients to clients if data replication is not complete. Faster data access is the primary concern, so we might opt for a less-strict consistency model **such as eventual consistency**.

Eventual Consistency

In this approach,the system informally ensures that, if no new updates are made to a particular piece of data, eventually all reads to that item will return the last updated value. The updated replicas send the update messages to all other replicas. In these states different replicas could return different values if queried, but eventually all the replicas get the update and will be consistent. This model is suitable for hundreds of thousands of concurrent reads are writes per second such as Twitter updates, Instagram photo uploads, Facebook status pages, messaging systems, and so on **where data integrity concern is not paramount**.

Read-Your-Write Consistency

RYW (Read-Your-Writes) consistency is achieved when the system guarantees that, once a record has been updated, any attempt to read the record will return the updated value. **RDBMS generally gives read-your-write consistency**.

Read-after-Write Consistency

Read-after-write consistency is stricter than eventual consistency. A newly inserted data item or record will be immediately visible to all the clients. **Please note that it is only applicable to new data. Updates and deletions are not considered in this model**.

Amazon S3 Consistency Models

Amazon S3 **provides read-after-write consistency for PUTS of new objects in your S3 bucket and eventual consistency for overwrite PUTS and DELETES in all regions**. So, if you add a new object to your bucket, you and your clients will see it. But, if you overwrite an object, it might take some time to update its replicas – hence the eventual consistency model is applied.

Amazon S3 guarantees high-availability by replicating data across many servers and AZs. It is obvious that data integrity should be maintained if a new record is added or a record/data is updated and deleted. The scenarios for above cases are as follows:

- **A new PUT request is made**. The object might not appear in the list if queried immediately until the changes are propagated to all the servers and AZs. The read-after-write consistency model is applied here.

- **An UPDATE request is made**. As eventual consistency model is applied for UPDATEs, a query to list the object might return an old value.
- **A DELETE request is made**. As eventual consistency model is applied for DELETEs, a query to list or read the object might return the deleted object.

Amazon DynamoDB Consistency Models

Amazon DynamoDB is one of the most popular NoSQL service from AWS. **NoSQL storage is inherently distributed**. To enable high availability and data durability, Amazon DynamoDB stores three geographically distributed replicas of each table. **A write operation in DynamoDB adheres to eventual consistency**. A read operation (*GetItem, BatchGetItem, Query or Scan operations*) on DyanamoDB table is eventual consistent read by default. But, you can configure a strong consistent read request for the most recent data. **Note that a strong consistent read operation consumes twice the read units than eventual consistent read request**. In general, it is advised to follow eventual consistent read because the change propagation in DynamoDB is very fast (DynamoDB uses SSDs for low-latency) and you will get the same result with the half of the cost of a strong read consistent request.

Conclusion

Phew! That was a lot of information. I hope you now have at least some idea about the different types of consistency models. AWS's distributed paradigm means its services have to adopt consistency models which best suits the performance and consistency of data or objects.

Want to learn more? Try Cloud Academy for free for 7-days. Here are a few courses and learning paths that might interest you: