Consistency Model

- A consistency model is contract between a distributed data store and processes, in which the processes agree to obey certain rules in contrast the store promises to work correctly.
- A consistency model basically refers to the degree of consistency that should be maintained for the shared memory data.
- If a system supports the stronger consistency model, then the weaker consistency model is automatically supported but the converse is not true.
- The types of consistency models are Data-Centric and client centric consistency models.

1.Data-Centric Consistency Models

A data store may be physically distributed across multiple machines. Each process that can access data from the store is assumed to have a local or nearby copy available of the entire store.

i.Strict Consistency model

- Any read on a data item X returns a value corresponding to the result of the most recent write on X
- This is the strongest form of memory coherence which has the most stringent consistency requirement.
- Strict consistency is the ideal model but it is impossible to implement in a distributed system. It is based on absolute global time or a global agreement on commitment of changes.

ii.Sequential Consistency

- Sequential consistency is an important data-centric consistency model which is a slightly weaker consistency model than strict consistency.
- A data store is said to be sequentially consistent if the result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process should appear in this sequence in a specified order.
- Example: Assume three operations read(R1), write(W1), read(R2) performed in an order on a memory address. Then (R1,W1,R2),(R1,R2,W1),(W1,R1,R2)(R2,W1,R1) are acceptable provided all processes see the same ordering.

iii.Linearizability

- It that is weaker than strict consistency, but stronger than sequential consistency.
- A data store is said to be linearizable when each operation is timestamped and the result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order
- The operations of each individual process appear in sequence order specified by its program.
- If tsOP1(x)< tsOP2(y), then operation OP1(x) should precede OP2(y) in this sequence.

iv.Causal Consistency

- It is a weaker model than sequential consistency.
- In Casual Consistency all processes see only those memory reference operations in the correct order that are potentially causally related.
- Memory reference operations which are not related may be seen by different processes in different order.
- A memory reference operation is said to be casually related to another memory reference operation if the first operation is influenced by the second operation.
- If a write(w2) operation is casually related to another write (w1) the acceptable order is (w1, w2).

v.FIFO Consistency

- It is weaker than causal consistency.
- This model ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed like a single process in a pipeline.
- This model is simple and easy to implement having good performance because processes are ready in the pipeline.
- Implementation is done by sequencing write operations performed at each node independently of the operations performed on other nodes.
- Example: If (w11) and (w12) are write operations performed by p1 in that order and (w21),(w22) by p2. A process p3 can see them as [(w11,w12),(w21,w2)] while p4 can view them as [(w21,w2),(w11,w12)].

vi.Weak consistency

- The basic idea behind the weak consistency model is enforcing consistency on a group of memory reference operations rather than individual operations.
- A Distributed Shared Memory system that supports the weak consistency model uses a special variable called a synchronization variable which is used to synchronize memory.
- When a process accesses a synchronization variable, the entire memory is synchronized by making visible the changes made to the memory to all other processes.

vii.Release Consistency

- Release consistency model tells whether a process is entering or exiting from a critical section so that the system performs either of the operations when a synchronization variable is accessed by a process.
- Two synchronization variables acquire and release are used instead of single synchronization variable. Acquire is used when process enters critical section and release is when it exits a critical section.
- Release consistency can be viewed as synchronization mechanism based on barriers instead of critical sections.

viii.Entry Consistency

- In entry consistency every shared data item is associated with a synchronization variable.
- In order to access consistent data, each synchronization variable must be explicitly acquired.

- Release consistency affects all shared data but entry consistency affects only those shared data associated with a synchronization variable.

2.Client-Centric Consistency Models

- Client-centric consistency models aim at providing a system wide view on a data store.
- This model concentrates on consistency from the perspective of a single mobile client.
- Client-centric consistency models are generally used for applications that lack simultaneous updates were most operations involve reading data.

i.Eventual Consistency

- In Systems that tolerate high degree of inconsistency, if no updates take place for a long time all replicas will gradually and eventually become consistent. This form of consistency is called eventual consistency.
- Eventual consistency only requires those updates that guarantee propagation to all replicas.
- Eventual consistent data stores work fine as long as clients always access the same replica.
- Write conflicts are often relatively easy to solve when assuming that only a small group of processes can perform updates. Eventual consistency is therefore often cheap to implement.

ii.Monotonic Reads Consistency

- A data store is said to provide monotonic-read consistency if a process reads the value of a data item x, any successive read operation on x by that process will always return that same value or a more recent value.
- A process has seen a value of x at time t, it will never see an older version of x at a later time.
- Example: A user can read incoming mail while moving. Each time the user connects to a different e-mail server, that server fetches all the updates from the server that the user previously visited. Monotonic Reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.

iii.Monotonic Writes

- A data store is said to be monotonic-write consistent if a write operation by a process on a data item x is completed before any successive write operation on X by the same process.
- A write operation on a copy of data item x is performed only if that copy has been brought up to date by means of any preceding write operations, which may have taken place on other copies of x.
- Example: Monotonic-write consistency guarantees that if an update is performed on a copy of Server S, all preceding updates will be performed first. The resulting server will then indeed become the most recent version and will include all updates that have led to previous versions of the server.

iv.Read Your Writes

- A data store is said to provide read-your-writes consistency if the effect of a write operation by a process on data item x will always be a successive read operation on x by the same process.
- A write operation is always completed before a successive read operation by the same process no matter where that read operation takes place.
- Example: Updating a Web page and guaranteeing that the Web browser shows the newest version instead of its cached copy.

v.Writes Follow Reads

- A data store is said to provide writes-follow-reads consistency if a process has write operation on a data item x following a previous read operation on x then it is guaranteed to take place on the same or a more recent value of x that was read.
- Any successive write operation by a process on a data item x will be performed on a copy of x that is up to date with the value most recently read by that process.
- Example: Suppose a user first reads an article A then posts a response B. By requiring writes-follow-reads consistency, B will be written to any copy only after A has been written.