CLOUD COMPUTING

Consistency Models

Dr. Prafullata Kiran Auradkar

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 3 of the course have been created by **Dr. H.L. Phalachandra** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

Consistency and Eventual Consistency

- The term ***consistency*** refers to the consistency of the values in different copies of the same data item in a replicated distributed system.
- This consistency can be lost when there is a network issue or there is a difference in the time taken to write into different copies of the same data.
- 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.
- There are applications and environments which need strong consistency i.e. values across copies to be the same, which will need to use different mechanisms to achieve the same. These mechanisms can lead to transactions slowing down and hence having an impact on the performance.
- A consistency model basically refers to the degree of consistency that should be maintained for the shared memory data

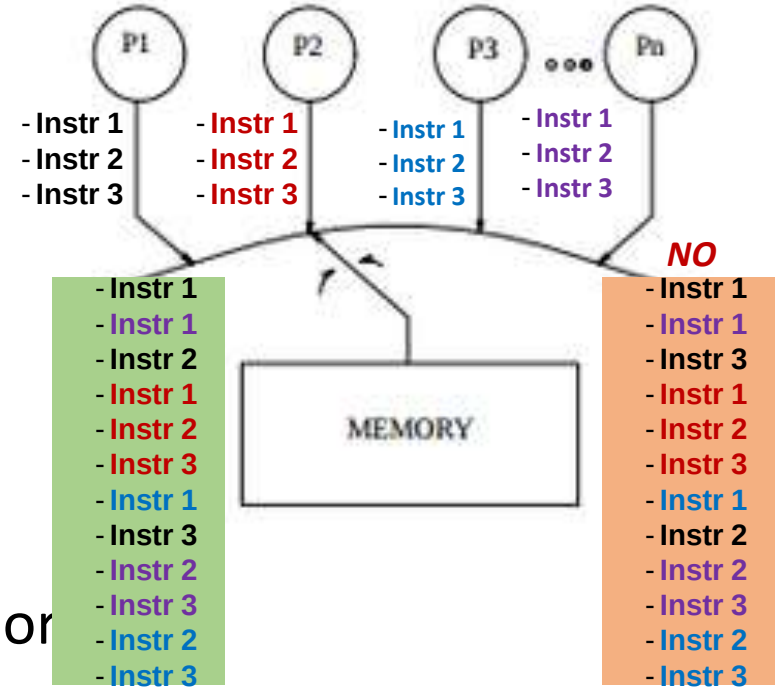
Consistency Models – Consistency Guarantees

- Eventual consistency is designing systems to eventually guarantee the copies of data to be consistent once all the current operations have been processed, but don't always have to be identical. This provides improved performance.
- Most replicated databases provide **eventual consistency**, which means that if you stop writing to the database and wait for some **unspecified length of time**, then eventually all read requests will return the same value. That is, all replicas will eventually converge to the same value
- This is a very **weak guarantee** as it doesn't say anything about when the replicas will converge
- The edge cases of eventual consistency only become apparent when there is a fault in the system or at high concurrency.

Consistency Models – Different Types

Sequential Consistency (Lamport)

- A shared-memory system is said to support the sequential consistency model, if all processes see the same order of all memory access operations on the shared memory.
- The exact order in which the memory access operations are interleaved does not matter... If one process sees one of the orderings of ... three operations and another process sees a different one, the memory is not a sequentially consistent memor



- Conceptually there is one global memory and a switch that connects an arbitrary processor to the memory at any time. Each processor issues memory operations in program order and the switch provides the global serialization among all the processors.
- Therefore: Sequential consistency is ***not deterministic*** because multiple execution of the distributed program might lead to a different order of operations.

Consistency Models – Different Types

Causal Consistency

- Two events are causally related if one can influence the other
- Relaxes the requirement of the sequential model for better concurrency. Unlike the sequential consistency model, in the causal consistency model, all processes see only those memory reference operations in the same (correct) order that are potentially causally related.
- Memory reference operations that are not potentially causally related may be seen by different processes in different orders.

PRAM (Pipelined Random-Access Memory) consistency

- It ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed as if all the write operations performed by a single process are in a pipeline.
- Write operations performed by different processes may be seen by different processes in different orders

Consistency Models – Different Types

Strict Consistency (also called Strong consistency or Linearizability)

- A shared-memory system is said to support the strict consistency model if the value returned by a read operation on a memory address is always the same as the value written by the most recent write operation to that address, irrespective of the locations of the processes performing the read and write operations
- Like sequential consistency, but the execution order of programs between processors must be the order in which those operations were issued.
- Therefore: If each program in each processor is deterministic, then the distributed program is deterministic.

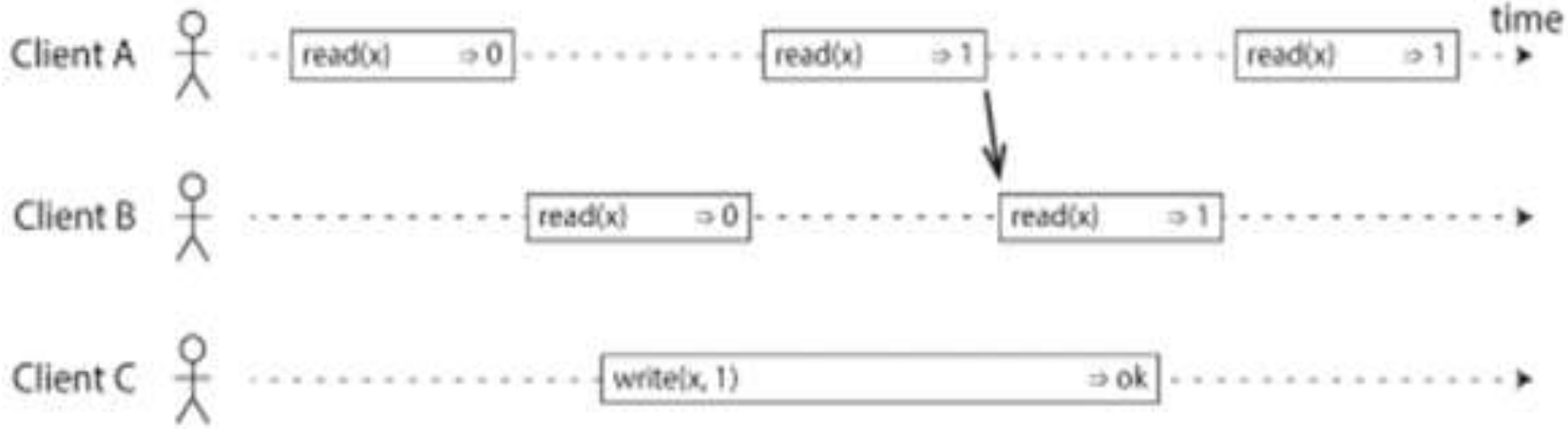
Linearizability

- Basic idea is to make a system appear as if there were only one copy of the data, and all operations on it are atomic.
- With this guarantee, even though there may be multiple replicas in reality, the application does not need to worry about them.
- Also known as atomic consistency, strong consistency, immediate consistency or external consistency
- **Linearizability** is a **recency guarantee**: a read is guaranteed to see the latest value written.
- In a linearizable system, as soon as one client successfully completes a write, all clients reading from the database must be able to see the value just written.
- Maintaining the illusion of a single copy of the data means guaranteeing that the value read is the most recent, up-to-date value and doesn't come from a stale cache or replica

Linearizability

- If a read request is concurrent with a write request, it may return either the old or the new value

- After any one read has returned the new value, all following reads (on the same or other clients) must also return the new value.



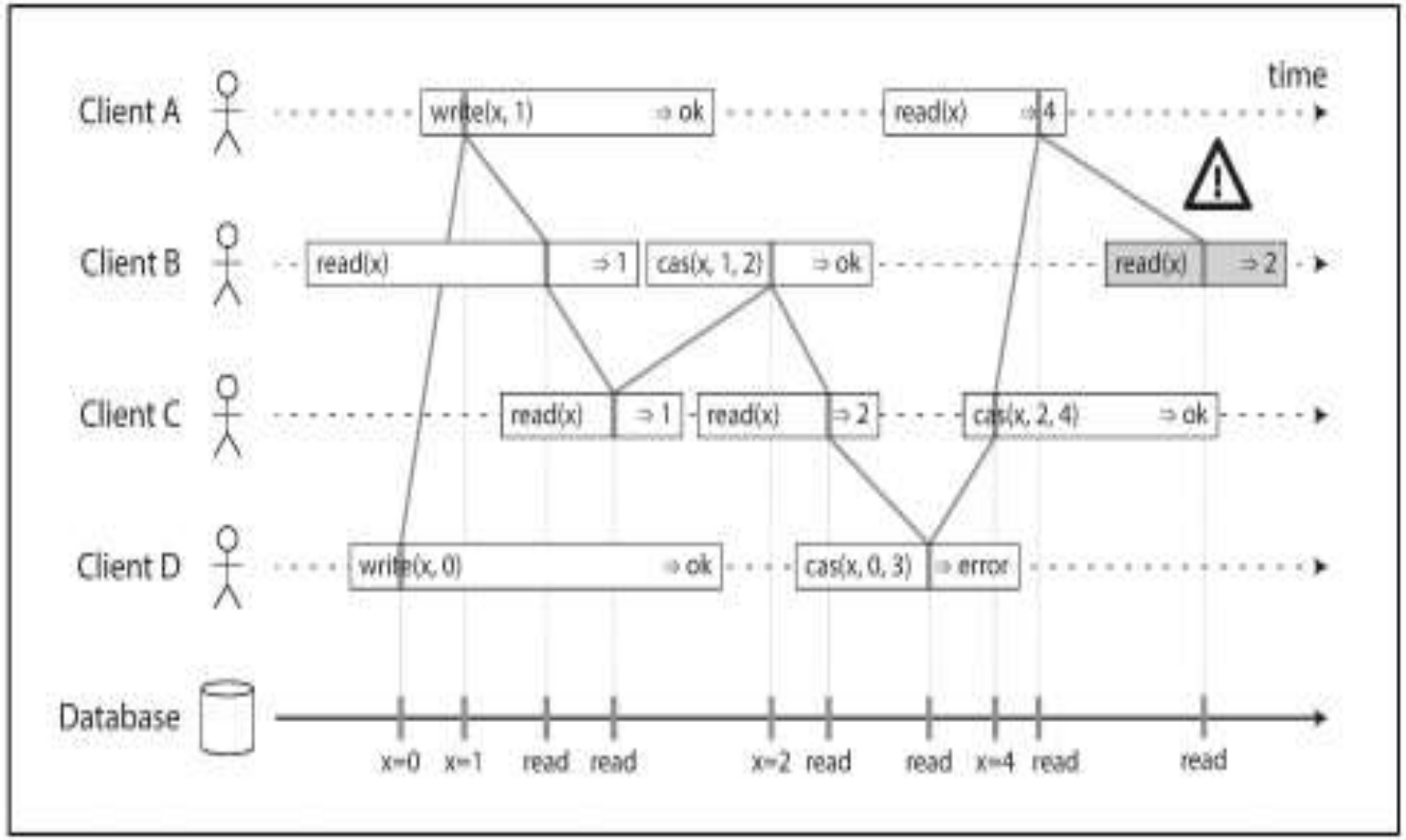
Compare and Set (cas)

- Add a third type of operation besides read and write
- **cas(x, v_{old}, v_{new}) ⇒ r** means the client requested an atomic compare-and-set operation . If the current value of the register x equals v_{old}, it should be atomically set to v_{new}. If $x \neq v_{old}$ then the operation should leave the register unchanged and return an error. r is the database's response (ok or error).

Linearizability

Visualizing the points in time at which the reads and writes appear to have taken effect

- The final read by B is not linearizable
- It is possible to test whether a system's behavior is linearizable by recording the timings of all requests and responses and checking whether they can be arranged into a valid sequential order



Linearizability

Single-leader replication (potentially linearizable)

- If you make reads from the leader or from synchronously updated followers, they have the potential to be linearizable

Consensus algorithms (linearizable)

- Consensus protocols contain measures to prevent split brain and stale replicas.
- Consensus algorithms can implement linearizable storage safely
- Example: ZooKeeper

Multi-leader replication (not linearizable)

Leaderless replication (probably not linearizable)

THANK YOU

Prafullata Kiran Auradkar

Department of Computer Science and Engineering

prafullatak@pes.edu