Consider that this is a distributed storage system that serves read/write requests.

- Multiple copies of a same object stored at different servers
- Question: How to maintain consistency across different data replicas?

### Consistency

- Why replicate?
  - Increased availability of service. When servers fail or when the network is partitioned.
    - \( P \): probability that one server fails \( \rightarrow 1 - P \approx \) availability of service. e.g. \( P = 5\% \Rightarrow \) service is available 95% of the time.
    - \( P^n \): probability that \( n \) servers fail \( \rightarrow 1 - P^n \approx \) availability of service. e.g. \( P = 5\% \), \( n = 3 \Rightarrow \) service available 99.875% of the time.
- Fault tolerance
  - Under the fail-stop model, if up to \( f \) of \( f+1 \) servers crash, at least one is alive.
- Load balancing
  - One approach: Multiple server IPs can be assigned to the same name in DNS, which returns answers round-robin.

### Our Expectation with Data

- Consider a single process using a filesystem
- What do you expect to read?

**P1**

\[ \text{x.write(2)} \quad \text{x.read()} \]

- Our expectation (as a user or a developer)
  - A read operation returns the most recent write.
  - This forms our basic expectation from any file or storage system.
- **Linearizability** meets this basic expectation.
  - But it extends the expectation to handle multiple processes...
  - ...and multiple replicas.
  - The strongest consistency model

**P2**

\[ \text{x.write(5)} \]

### Expectation with Multiple Processes

- What do you expect to read?
  - A single filesystem with multiple processes

**P1**

\[ \text{x.write(5)} \quad \text{x.read()} \]

- Our expectation (as a user or a developer)
  - A read operation returns the most recent write, regardless of the clients.
  - We expect that a read operation returns the most recent write according to the single actual-time order.
  - In other words, read/write should behave as if there were a single (combined) client making all the requests.
  - It's easiest to understand and program for a developer if your storage appears to process one request at a time.
Expectation with Multiple Copies

- What do you expect to read?
  - A single process with multiple servers with copies

  \[
  \text{P1: x.write(2) \ x.read()}?
  \]

- Our expectation (as a user or a developer)
  - A read operation returns the most recent write, \text{regardless of how many copies there are.}
  - Read/write should behave as if there were a single copy.

Linearizability

- Three aspects
  - A read operation returns the most recent write,
  - \text{regardless of the clients,}
  - \text{regardless of how many copies there are}

- Or, put it differently, \text{read/write should behave as if there were,}
  - \text{a single client making all the (combined) requests in their original actual-time order (i.e., with a single stream of ops),}
  - \text{over a single copy.}

- You can say that \text{your storage system guarantees linearizability when it provides single-client, single-copy semantics where a read returns the most recent write.}
  - \text{It should appear to all clients that there is a single order (actual-time order) that your storage uses to process all requests.}

Linearizability Exercise

- Assume that the following happened with object \(x\) over a linearizable storage.
  - C1: \(x.write(A)\)
  - C2: \(x.write(B)\)
  - C3: \(x.read() \rightarrow B, x.read() \rightarrow A\)
  - C4: \(x.read() \rightarrow B, x.read() \rightarrow A\)

- What would be an actual-time ordering of the events?
  - One possibility: \(C2 \ (write \ B) \rightarrow C3 \ (read \ B) \rightarrow C4 \ (read \ B) \rightarrow C1 \ (write \ A) \rightarrow C3 \ (read \ A) \rightarrow C4 \ (read \ A)\)

- How about the following?
  - C1: \(x.write(A)\)
  - C2: \(x.write(B)\)
  - C3: \(x.read() \rightarrow B, x.read() \rightarrow A\)
  - C4: \(x.read() \rightarrow A, x.read() \rightarrow B\)

Linearizability Subtleties

- Notice any problem with the representation?

  
  \[
  \text{You (NY): x.write(5) \ x.read()}?
  \]

  \[
  \text{Friend (CA): x.write(2) \ x.read()}?
  \]

CSE 486/586 Administrivia

- PA4 deadline: 5/10 (Friday)
- No recitation today
  - Will hold office hours for undergrads from 2:30 pm to 4 pm
  - Regular office hours from 4 pm to 5 pm for midterm questions
- Academic integrity for PA2B

Linearizability Subtleties

- A read/write operation is never a dot!
  - It takes time. Many things are involved, e.g., network, multiple disks, etc.
  - Read/write latency: the time measured right before the call and right after the call from the client making the call.

- Clear-cut (e.g., black---write & red---read)

- Not-so-clear-cut (parallel)
  - Case 1:
  - Case 2:
  - Case 3:
Linearizability Subtleties

• With a single process and a single copy, can overlaps happen?
  – No, these are cases that do not arise with a single process and a single copy.
  – "Most recent write" becomes unclear when there are overlapping operations.

• Thus, we (as a system designer) have freedom to impose an order.
  – As long as it appears to all clients that there is a single, interleaved ordering for all (overlapping and non-overlapping) operations that your implementation uses to process all requests, it’s fine.
  – I.e., this ordering should still provide the single-client, single-copy semantics.
  – Again, it’s all about how clients perceive the behavior of your system.

Definite guarantee

• Relaxed guarantee when overlap
  • Case 1
  • Case 2
  • Case 3

Examples

• Example 1: if your system behaves this way with 3 clients…

  a.write(x)
  a.read() -> x
  a.read() -> x
  a.read() -> x

• Example 2: if your system behaves this way with 3 clients…

  a.write(x)
  a.read() -> 0
  a.read() -> x
  a.read() -> x

If this were a.read() -> 0, would it support linearizability?
No

• In example 2, what are the constraints?

  a.write(x)
  a.read() -> 0
  a.read() -> x
  a.read() -> x

• Constraints (some ops don’t overlap)
  – a.read() -> 0 happens before a.read() -> x (you need to be able to explain why that happens that way).
  – a.read() -> x happens before a.read() -> x (you need to be able to explain why that happens that way).
  – The rest are up for grabs.

• Scenario
  – Every client deals with a different copy of a.
  – a.write(x) gets propagated to (last client’s) a.read() -> x first.
  – a.write(x) gets propagated to (the second process’s) a.read() -> x, right after a.read() -> 0 is done.

• Example 3

  a.write(x)
  a.read() -> x
  a.read() -> x
  a.read() -> y
  a.write(y)

• Constraints (ops that don’t overlap)
  – a.read() -> x and a.read() -> x: we cannot change these.
  – a.read() -> y and a.read() -> x: we cannot change these.
  – The rest is up for grabs.
**Linearizability (Textbook Definition)**

- Let the sequence of read and update operations that client \( i \) performs in some execution be \( \alpha_1, \alpha_2, \ldots \).
  - “Program order” for the client
- A replicated shared object service is *linearizable* if for any execution (real), there is some interleaving of operations (virtual) issued by all clients that:
  - meets the specification of a single correct copy of objects
  - is consistent with the actual times at which each operation occurred during the execution
- Main goal: any client will see (at any point of time) a copy of the object that is correct and consistent
- The strongest form of consistency

**Summary**

- Linearizability
  - Single-client, Single-copy semantics
- A read operation returns the *most recent* write, regardless of the clients, according to their actual-time ordering.

**Acknowledgements**

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).