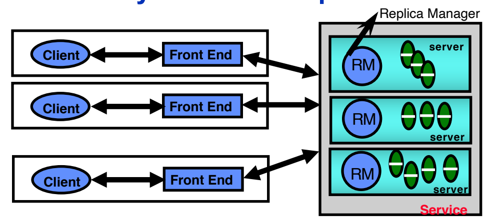
CSE 486/586 Distributed Systems
Consistency --- 1

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Consistency with Data Replicas



- 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 → 1 – P= availability of service. e.g. P = 5% => service is available 95% of the time.
  - P^n: probability that n servers fail → 1 – P^n= availability of service. e.g. P = 5%, n = 3 => service available 99.9875% 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.

This Week

- We will look at different consistency guarantees (models).
- We'll start from the strongest guarantee, and gradually relax the guarantees.
  - Linearizability (or sometimes called strong consistency)
  - Sequential consistency
  - Causal consistency
  - Eventual consistency
- Different applications need different consistency guarantees.
- This is all about client-side perception.
  - When a read occurs, what do you return?
- First
  - Linearizability: we'll look at the concept first, then how to implement it later.

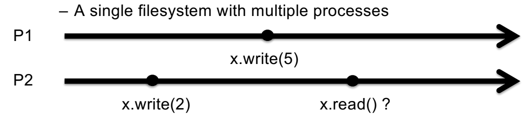
Our Expectation with Data

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

- Our expectation (as a user or a developer)
  - A read operation would return 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

Expectation with Multiple Processes

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



- Our expectation (as a user or a developer)
  - A read operation would return the most recent write, regardless of the clients.
  - We expect that a read operation would return the most recent write according to the single physical-time order.
  - In other words, we expect read/write to behave as if there were a single (combined) client making all the requests.

Expectation with Multiple Copies

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

P1 —— x.write(2) —— x.read() ? ——→

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

Linearizability

- Three aspects
  – A read operation should return the most recent write according to physical time,
  – …regardless of how many clients there are,
  – …and regardless of how many copies there are.
- Or, put it differently, read/write should behave as if there were,
  – …a single client making all the (combined) requests in their original physical-time order,
  – …over a single copy.
  – This is called the single-client, single-copy semantics.
- You can say that your storage system guarantees linearizability when it provides the above behavior.

Linearizability Exercise

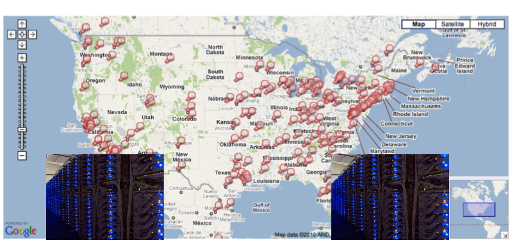
- Assume that the following happened with object x over a linearizable storage.
  – C1: x.write(A)
  – C2: x.write(B)
  – C3: x.read() → B, x.read() → A
  – C4: x.read() → B, x.read() → A
- What would be a physical-time ordering of the events?
  – One possibility: C2 (write B) -> C3 (read B) -> C4 (read B) -> C1 (write A) -> C3 (read A) -> C4 (read A)
- How about the following?
  – C1: x.write(A)
  – C2: x.write(B)
  – C3: x.read() → B, x.read() → A
  – C4: x.read() → A, x.read() → B

CSE 486/586 Administrivia

Linearizability Subtleties

- Notice any problem with the representation?

You (NY) —— x.write(5) ——→
Friend (CA) —— x.write(2) —— read(x) ? ——→



California North Carolina

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.
  - Still, linearizability requires your system to behave like it had a single client and a single copy.
- Thus, you (as a system designer) need to pick an ordering to process overlapping operations.
  - This ordering should still satisfy single-client, single-copy semantics.
  - In other words, given a read/write behavior of your system, you should be able to answer the following question: "what is your processing order that behaves like a single client issuing requests over a single copy?"

Linearizability Subtleties

- Definite guarantee
- Relaxed guarantee when overlap
- Case 1
- Case 2
- Case 3

Linearizability Examples

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

a.write(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

Linearizability Examples

- In example 2, what are the constraints?

a.write(x)

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

a.read() -> x

- Ordering constraints (non-overlapping ops)
  - a.read() → 0 happens before a.read() →x
  - a.read() → x happens before a.read() →x
  - In whatever ordering you use to explain the behavior, you can't change these two orderings.
- 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 client's) a.read() -> x, right after a.read() -> 0 is done.

Linearizability Examples

- Example 3

a.write(x)

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

a.read() -> y

a.write(y)

- Ordering constraints (ops that don't overlap)
  - a.read() → x and a.read() → x
  - a.read() → y and a.read() → x

Linearizability (Textbook Definition)

- Let the sequence of read and update operations that client i performs in some execution be oi1, oi2,….
  - "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 how many clients there are, regardless how many copies there are, and according to their physical-time ordering.

Acknowledgements

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