CSE 486/586 Distributed Systems
Consistency --- 2

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Recap: Linearizability

- Linearizability
  - Should provide the behavior of a single client and a single copy
  - A read operation returns the most recent write, regardless of the clients according to their original actual-time order.
- Complication
  - In the presence of concurrency, read/write operations overlap.

Linearizability Examples

- Example 1

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

- Example 2

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

If this were a.read() -> 0, it wouldn't support linearizability.

Linearizability Examples

- Example 3

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

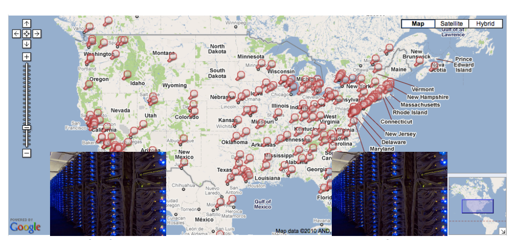
Linearizability

- Linearizability is all about client-side perception.
  - The same goes for all consistency models for that matter.
- If you write a program that works with a linearizable storage, *it works as you expect it to work*.
- There's no surprise.

Implementing Linearizability

- Will this be difficult to implement? Any strategy?

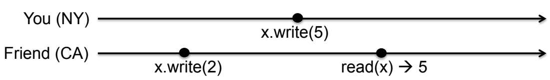
You (NY) — x.write(5)
Friend (CA) — x.write(2) — read(x) → 5



California North Carolina

Implementing Linearizability

- Will this be difficult to implement?
  - It depends on what you want to provide.

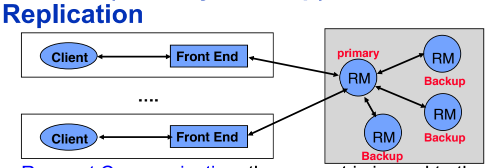


- How about:
  - All clients send all read/write to CA datacenter.
  - CA datacenter propagates to NC datacenter.
  - A request never returns until all propagation is done.
  - Correctness (linearizability)? yes
  - Performance? No

Implementing Linearizability

- Importance of latency
  - Amazon: every 100ms of latency costs them 1% in sales.
  - Google: an extra .5 seconds in search page generation time dropped traffic by 20%.
- Linearizability typically requires *complete* synchronization of multiple copies before a write operation returns.
  - So that any read over any copy can return the most recent write.
  - No room for asynchronous writes (i.e., a write operation returns before all updates are propagated.)
- It makes less sense in a global setting.
  - Inter-datecenter latency: ~10s ms to ~100s ms
- It still makes sense in a local setting (e.g., within a single data center).

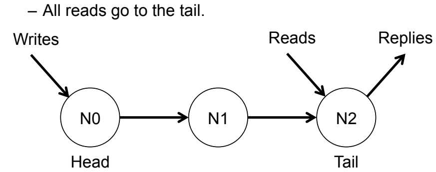
Passive (Primary-Backup) Replication



- Request Communication: the request is issued to the primary RM and carries a unique request id.
- Coordination: Primary takes requests atomically, in order, checks id (resends response if not new id.)
- Execution: Primary executes & stores the response
- Agreement: If update, primary sends updated state/ result, req-id and response to all backup RMs (1-phase commit enough).
- Response: primary sends result to the front end

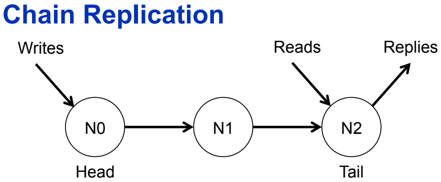
Chain Replication

- One technique to provide linearizability with better performance
  - All writes go to the head.
  - All reads go to the tail.



- Linearizability?
  - Clear-cut cases: straightforward
  - Overlapping ops?

Chain Replication



- What ordering does this have for overlapping ops?
  - We have freedom to impose an order.
  - Case 1: A write is at either N0 or N1, and a read is at N2. The ordering we're imposing is read then write.
  - Case 2: A write is at N2 and a read is also at N2. The ordering we're imposing is write then read.
- Linearizability
  - Once a write becomes visible (at the tail), all following reads get the write result.

CSE 486/586 Administrivia

- PA3 deadline: 4/3 (Friday)

Relaxing the Guarantees

- Do we need linearizability?

- Does it matter if I see some posts some time later?
- Does everyone need to see these in this particular order?

Relaxing the Guarantees

- Linearizability advantages
  - It behaves as expected.
  - There's really no surprise.
  - Application developers do not need any additional logic.
- Linearizability disadvantages
  - It's difficult to provide high-performance (low latency).
  - It might be more than what is necessary.
- Relaxed consistency guarantees
  - Sequential consistency
  - Causal consistency
  - Eventual consistency
- It is still all about client-side perception.
  - When a read occurs, what do you return?

Sequential Consistency

- A little weaker than linearizability, but still quite strong
- Consider the same scenario & our expectation.

P1: x.write(2) x.read() → 2

- What about the following?

P1: x.write(5)

P2: x.write(2) x.read() → 2 x.read() → 5

Sequential Consistency

- What about this?

P1: x.write(5)

P2: x.write(2) x.write(3) x.read() → 3 x.read() → 2

- And this?

P1: x.write(5) x.write(3)

P2: x.write(2) x.read() → 5 x.read() → 3

Sequential Consistency

P1: x.write(5) x.write(3)

P2: x.write(2) x.read() → 2 x.read() → 5 x.read() → 3

- Observation: It's *still reasonable* (for many apps),
  - …to not strictly follow the actual-time ordering across clients,
  - …as long as it preserves the program order of each client.
- This meets the expectation from a (isolated) client.
  - Linearizability meets the expectation of all clients in a global sense.

Sequential Consistency

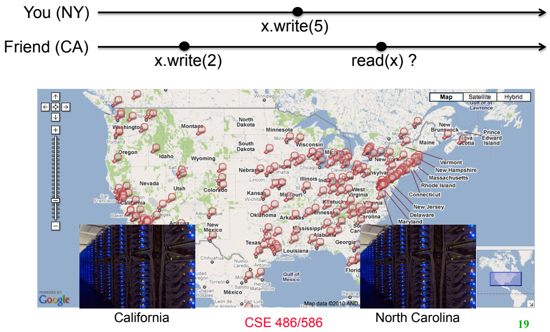
- Similar to linearizability, and it should behave as if there were only a single copy, and a single client.
  - It's just that it doesn't preserve the actual-time order, but just the program order of each client.
- Difference

You (NY): x.write(5)

Friend (CA): x.write(2) read(x) ?

  - Linearizability: Once a write is returned, the system is obligated to make the result visible to all clients based on actual time. I.e., the system has to return 5 in the example.
  - Sequential consistency: Even if a write is returned, the system is not obligated to make the result visible to other clients immediately. I.e., the system can still return 2 in the example.

Sequential Consistency



Sequential Consistency

- Read/write should behave as if there were,
  - …a single client making all the (combined) requests *not in their original actual-time order* but in an interleaving that preserves the program order of each client,
  - …over a single copy.
- Both linearizability and sequential consistency care about giving an illusion of a single copy.
  - From the outside observer, the system should behave as if there were only a single copy.

Sequential Consistency Examples

- Example 1: Can a sequentially consistent storage show this behavior?
  - P1: a.write(A)
  - P2: a.write(B)
  - P3: a.read()->B a.read()->A
  - P4: a.read()->B a.read()->A
- Example 2
  - P1: a.write(A)
  - P2: a.write(B)
  - P3: a.read()->B a.read()->A
  - P4: a.read()->A a.read()->B

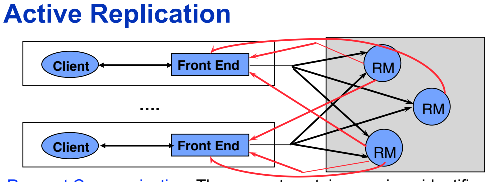
Implementing Sequential Consistency

- In what implementation would the following happen?
  - P1: a.write(A)
  - P2: a.write(B)
  - P3: a.read()->B a.read()->A
  - P4: a.read()->A a.read()->B
- Possibility
  - P3 and P4 use different copies.
  - In P3's copy, P2's write arrives first and gets applied.
  - In P4's copy, P1's write arrives first and gets applied.
  - Writes are applied in different orders across copies.
  - This doesn't provide sequential consistency.

Implementing Sequential Consistency

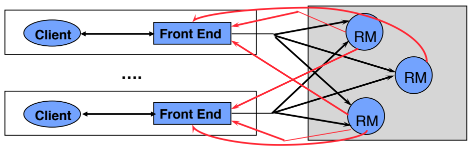
- Like linearizability:
  - Write synchronization needs to happen in the same order everywhere across different copies.
  - I.e., writes should be applied in the same order across different copies.
  - Otherwise, it cannot behave as if there were a single copy.
- Different from linearizability:
  - The synchronization does not have to be complete at the time of return from a write operation.
- Typical implementation
  - You're not obligated to make the most recent write (according to actual time) visible (i.e., applied to all copies) right away.
  - But you are obligated to apply all writes in the same order for all copies. This order should be FIFO-total.

Active Replication



- Request Communication: The request contains a unique identifier and is multicast to all by a reliable totally-ordered multicast.
- Coordination: Group communication ensures that requests are delivered to each RM in the same order.
- Execution: Each replica executes the request. (Correct replicas return same result since they are running the same program, i.e., they are replicated protocols or replicated state machines)
- Agreement: No agreement phase is needed, because of multicast delivery semantics of requests
- Response: Each replica sends response directly to FE

Active Replication



- A front end FIFO-orders all reads and writes.
- A read can be done completely with any replica.
- Writes are totally-ordered and asynchronous (after at least one write completes, it returns).
  - Total ordering doesn't guarantee when to deliver events, i.e., writes can happen at different times at different replicas.
- Sequential consistency, not linearizability
  - Read/write ops from the same client will be ordered at the front end (program order preservation).
  - Writes are applied in the same order by total ordering (single copy).
  - No guarantee that a read will read the most recent write based on actual time.

Two More Consistency Models

- Even more relaxed
  - We don't even care about providing an illusion of a single copy.
- Causal consistency
  - We care about ordering causally related write operations correctly.
- Eventual consistency
  - As long as we can say all replicas converge to the same copy eventually, we're fine.

Summary

- Linearizability
  - The ordering of operations is determined by time.
  - Primary-backup can provide linearizability.
  - Chain replication can also provide linearizability.
- Sequential consistency
  - The ordering of operations preserves the program order of each client.
  - Active replication can provide sequential consistency.

Acknowledgements

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