Recap: Concurrency (Transactions)

- Question: How to support transactions (with locks)?
  - Multiple transactions share data.
- First strategy: Complete serialization
  - One transaction at a time with one big lock
  - Correct, but at the cost of performance
- How to improve performance?
  - Let’s see if we can concurrently execute transactions.

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?

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 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

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

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

Linearizability
- Three aspects
  - A read operation returns the most recent write,
  - ...regardless of the clients,
  - ...according to the single actual-time ordering of requests.
- Or, put it differently, read/write should behave as if there were,
  - ...a single client making all the (combined) requests in their original actual-time order,
  - ...over a single copy.
- You can say that your storage system guarantees linearizability when it provides single-client, single-copy semantics where a read returns the most recent write.

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

CSE 486/586 Administrivia
- PA3 deadline: 4/3 (Friday)
- Grading is going on with PA2B and midterm.
Linearizability Subtleties

• Notice any problem?

You (NY)  
Friend (CA)  

x.write(5)  
x.write(2)  
read(x) ?

• 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: 

• Definite guarantee

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

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

  • Constraints
    – a.read() -> 0 happens before a.read() -> x (cannot change the order).
    – a.read() -> x happens before a.read() -> x (cannot change the order).
    – The rest are up for grabs.
Linearizability Examples

- In example 2, why would `a.read()` return 0 and `x` when they're overlapping?
  
  ```
  a.write(x)
  a.read() -> 0  a.read() -> x
  a.read() -> x
  ```

- This assumes that there's a particular storage system that shows this behavior.
- At some point between a read/write request sent and returned, the result becomes visible.
  
  - E.g., you read a value from physical storage, prepare it for return (e.g., putting it in a return packet, i.e., making it visible), and actually return it.
  - Or you actually write a value to a physical disk, making it visible (out of multiple disks, which might actually write at different points).

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 the clients, according to their actual-time ordering.

Acknowledgements

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