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

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-datacenter latency: ~10s ms to ~100s ms
– It might still make 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

N0 \rightarrow N1 \rightarrow N2

- 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
- PA4 deadline: 5/10 (Friday)
- 486/586 survey

Relaxing the Guarantees
- Do we need linearizability?
- 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.

Sequential Consistency
- A little weaker than linearizability, but still quite strong
  - Essentially linearizability, except that it doesn't need to return the most recent write according to physical time.

Delayed Write Visibility
- Consider the following:
  - Let's assume P1 and P2 are two users separately using Facebook. x is the same wall.
    - The first read at P2 is reading P2's own wall post.
    - The second read at P2 is reading P1's wall post at a delayed point in time, not right away.
    - With linearizability, P2's first read should read P1's wall post (the most recent write).
    - But again do we always need to? For applications like Facebook, delayed write visibility is probably fine.
Delayed Write Visibility

- Consider the following single process.

  \[
  \text{P1: } x.\text{write}(2) \quad x.\text{write}(3) \quad x.\text{read()} \rightarrow ?
  \]

- If delayed write visibility is fine, is it okay for \( x.\text{read()} \) to return 2, not 3?
- If not, why not?
  - Now we're violating the program order for a single process.
  - Developers will start getting really confused.
  - With a single copy, that will never happen.

Sequential Consistency Definition

- Insight
  - For many applications, we don't need to make other processes' writes immediately visible, as long as we preserve each process's program order.
  - Still a strong guarantee: If a process doesn't know what other processes are doing (e.g., when other processes' writes are occurring), this still meets the natural expectation of the process.
  - You can say that your storage system provides sequential consistency if:
    - All requests appear to come from a single client with a single interleaving of all requests.
    - In the single interleaving, the program order of each and every process is preserved.
  - This still works like a single copy, but all program orders are only logically preserved.

Sequential Consistency Examples

- Example 1: Does this satisfy sequential consistency?
  - No: even if P1's writes show up later, we can't explain the last two writes.

- Example 2: Does this satisfy sequential consistency?
  - Yes

- Example 3
  - P1: \( a.\text{write}(A) \)
  - P2: \( a.\text{write}(B) \)
  - P3: \( a.\text{read}() \rightarrow B \quad a.\text{read}() \rightarrow A \)
  - P4: \( a.\text{read}() \rightarrow B \quad a.\text{read}() \rightarrow A \)

- Example 4
  - P1: \( a.\text{write}(A) \)
  - P2: \( a.\text{write}(B) \)
  - P3: \( a.\text{read}() \rightarrow B \quad a.\text{read}() \rightarrow A \)
  - P4: \( a.\text{read}() \rightarrow A \quad a.\text{read}() \rightarrow B \)
Sequential Consistency

- Your storage appears to process all requests in a single interleaved ordering (single client), where...
  - ...each and every process’s program order is preserved (single copy),
  - ...and each process’s program order is only logically preserved, i.e., it doesn’t need to preserve its physical-time ordering.
- It works as if all clients are reading out of a single copy.
  - This meets the expectation from an (isolated) client, working with a single copy.
  - Linearizability meets the expectation of all clients even if they all know what others are doing.
  - Both sequential consistency and linearizability provide an illusion of a single copy.

Sequential Consistency vs. Linearizability

- Both should behave as if there were only a single copy, and a single client.
  - It’s just that SC doesn’t preserve the physical-time order, but just the program order of each client.
- Difference
  - Linearizability: Once a write is returned, the system is obligated to make the result visible to all clients based on physical 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.

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

- Typical implementation
  - You’re not obligated to make the most recent write (according to physical 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.
- What is this ordering guarantee?
  - FIFO-total.

Active Replication

- A front end FIFO-orders all reads and writes.
- A read can be done completely with any single replica.
- Writes are totally-ordered and asynchronous (after at least one write completes, it returns).
  - Total ordering doesn’t determine deliver times, 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 physical 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).