Implementing Linearizability

• Will this be difficult to implement?
  – It depends on what you want to provide.
  
  [Diagram showing two processes: You (NY) and Friend (CA). You write to process 5, Friend writes to process 2, and both processes read from process 5.]

• 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-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.

  [Diagram showing a chain replication with N0, N1, N2, and N3. N0 is the head, N1 and N2 are intermediate nodes, and N3 is the tail. All writes are directed to the head, and all reads are directed 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.

**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?

**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.
- How can we achieve it?
  - Preserving the single-client, (per-process) single-copy semantics
  - We give an illusion that there's a single copy to an isolated process.
- The single-client semantics
  - Processing all requests as if they were coming from a single client (in a single stream of ops).
  - Again, this meets our basic expectation—it's easiest to understand for an app developer if all requests appear to be processed one at a time.
- Let's consider the per-process single-copy semantics with a few examples.

**Per-Process Single-Copy Semantics**
- Consider the following single process.
- What do you expect to read?
  - 3, not 2
  - Why even consider 2? E.g., if there were two copies not synchronized correctly, two writes could be applied to different copies.
- Why 3 then?
  - It's the program order.
- Per-process single-copy semantics
  - When a storage system preserves a process's program order, the process will believe that there's a single copy.
**Per-Process Single-Copy Semantics**
- But we need to make it work with multiple processes.
  - When a storage system preserves each and every process’s program order, each will think that there’s a single copy.
- **Simple example**
  - P1: x.write(2), x.write(3), x.read() → 3
  - P2: x.write(5), x.read() → 5
- **Per-process single-copy semantics**
  - A storage system preserves each and every process’s program order.
  - It gives an illusion to every process that they’re working with a single copy.

**Pre-Process Single-Copy Examples**
- Example 1: Does this work like a single copy at P2?
  - P1: x.write(5)
  - P2: x.write(2), x.write(3), x.read() → 3, x.read() → 3
  - Yes!
  - Does this satisfy linearizability?
    - Yes

**Sequential Consistency**
- Insight: we don’t need to make other processes’ writes immediately visible.
- Central question
  - Can you explain a storage system’s behavior by coming up with a single interleaving ordering of all requests, where the program order of each and every process is preserved?
- **Previous example**
  - P1: x.write(5)
  - P2: x.write(2), x.write(3), x.read() → 3, x.read() → 5
  - We can explain this behavior by the following ordering of requests
    - x.write(2), x.write(3), x.read() → 3, x.write(5), x.read() → 5
- Example 1: Does this satisfy sequential consistency?
  - No: even if P1’s writes show up later, we can’t explain the last two writes.
Sequential Consistency Examples

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

```
P1: x.write(2) x.write(3) x.read() \rightarrow 3
P2: x.write(5) x.read() \rightarrow 5
```

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() \rightarrow B a.read() \rightarrow A
  - P4: a.read() \rightarrow A a.read() \rightarrow 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 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).