Introduction to Consistency

A distributed system may store data in multiple places:

- to improve performance
- to increase availability
- to provide fault tolerance

Multiple copies of the same object are called replicas.

For some applications, it may be important that these copies are maintain some form of consistency.

For example:

- they are the same
- they share history up to some point
- if they are different, one happens-before the other
- etc.
Advantages of Replication I

Replication can be used to improve performance.
For example:

- Placing replicas “near” clients on the network can improve network latency and bandwidth. (This is the foundation of content distribution networks like Akamai.)

- Distributing load between replicas (load balancing) can reduce individual server loads and improve response times.
Advantages of Replication II

It can also be used to increase availability.

Assume that the probability of a single server failing is $P$.
- With one server, expected availability is $1 - P$.
- With $n$ servers, expected availability is $1 - P^n$.

Thus, with $P = 5\%$:

<table>
<thead>
<tr>
<th>Servers</th>
<th>Availability (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>95.0000</td>
</tr>
<tr>
<td>2</td>
<td>99.7500</td>
</tr>
<tr>
<td>3</td>
<td>99.9875</td>
</tr>
<tr>
<td>4</td>
<td>99.9994</td>
</tr>
</tbody>
</table>
Consistency Guarantees

We will look at consistency guarantees in decreasing strength:

- Linearizability (or strong consistency)
- Sequential consistency
- Causal consistency
- Eventual consistency

The consistency required varies with the application.
Expectations within a Single Process

Consider a single process and a single datum:

```
write(x, 2);
y = read(x);
```

What do we expect y to contain?
Consider a single process and a single datum:

\[
\begin{align*}
\text{write}(x, 2); \\
y = \text{read}(x);
\end{align*}
\]

What do we expect \( y \) to contain?

Our general expectation is that

\text{a read operation returns the most recent write.}
Expectations with Multiple Processes

Consider a single datum across two processes:

Process P1:
write(x, 2);
y = read(x)

Process P2:
write(x, 2);

What do we expect y to contain?
Expectations with Multiple Processes

Consider a single datum across two processes:

Process P1:
write(x, 2);
y = read(x)

Process P2:
write(x, 2);

What do we expect \( y \) to contain?

- A read operation returns the most recent write regardless of who wrote.
- That most recent write is in physical time order.
- In other words, as if all reads and writes are serialized.
Expectation with Multiple Replicas

Similarly, with multiple replicas, we expect:

- Reads will return the most recent write regardless of the number of replicas.
- Read from any replica should return the same value.
- Write to any replica should modify all replicas.

Read and write operations should behave as if there were one replica.
Linearizability meets the foregoing expectations with the following guarantees:

- A read operation returns the most recent write,
- regardless of the number of clients, and
- according to the physical time ordering of requests.

This is equivalent to saying that linearization behaves as if a single client executed all operations in their physical time order on a single copy of the data.

A storage system guarantees linearizability if it provides the above single-client, single-copy semantics, such that a read returns the most recent write in physical time order.
Linearizing Real Operations I

Unfortunately, real operations do not occur instantaneously.

For example, disk and network I/O have latency.

Read/write latency: The time from when a call for a read or write is invoked until the time the operation returns control to the client.
Linearizing Real Operations II

Clear ordering:

```
Time
1. 
2. 
3. 
```

Unclear ordering:

```
Time
1. 
2. 
3. 
```

(write, read)

With multiple processes and operations of measurable length, overlaps can occur.

In this case, what is the correct order?

- It’s not clear!
Linearizing Real Operations III

Can overlaps happen with a single process and a single replica?

- No.
- Why not?

Linearizability picks something and defines it as correct:

- When overlaps occur, if it appears to all clients that there is a single, interleaved ordering for all operations, the ordering is valid.
- Once this ordering is defined, the correct value of each read is clear.

This means that there may be more than one valid linearization with overlap, but not within any one system.
Linearizability with Overlap

Consider three processes performing operations:

```
write(a, x)
read(a) → 0  read(a) → x
read(a) → x
```

Time

What are the constraints here?
Linearizability with Overlap

Consider three processes performing operations:

<table>
<thead>
<tr>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>write(a, x)</td>
</tr>
<tr>
<td>read(a) → 0</td>
</tr>
<tr>
<td>read(a) → x</td>
</tr>
</tbody>
</table>

What are the constraints here?

- read(a) → 0 happens before either read(a) → x
- write(a, x) happens before either read(a) → x
- The rest of the ordering can be implementation-defined
Linearizability with Overlap Analysis

Consider three processes performing operations:

<table>
<thead>
<tr>
<th>Time</th>
<th>write(a, x)</th>
<th>read(a) → 0</th>
<th>read(a) → x</th>
<th>read(a) → x</th>
</tr>
</thead>
</table>

How can this happen?

- The write(a, x) propagates to the bottom process first.
- The write(a, x) propagates to the middle process only after its read(a) → 0 has determined a value for a.
Linearizability with Overlap Analysis

Consider three processes performing operations:

\[
\begin{array}{c}
\text{write}(a, x) \\
\text{read}(a) \rightarrow 0 \\
\text{read}(a) \rightarrow x \\
\text{read}(a) \rightarrow x \\
\end{array}
\]

Time

Why might the middle process read different values for a \textit{during} the top process’s write?
Linearizability with Overlap Analysis

Consider three processes performing operations:

<table>
<thead>
<tr>
<th>Time</th>
<th>write(a, x)</th>
<th>read(a) → 0</th>
<th>read(a) → x</th>
<th>read(a) → x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Why might the middle process read different values for a during the top process’s write?

- At some point, the write becomes visible to other processes. (e.g., the value is actually stored to disk).
- The same is true for a read; at some point the read value is determined from the underlying storage.
Linearizability (Textbook Definition)

- Let the sequence of read and update operations that client \( i \) performs in some execution be \( o_{i1}, o_{i2}, \ldots \)

- A replicated shared object service is linearizable if, for any execution (in physical time), there is some interleaving of operations issued by all clients that:
  - meets the specification of a single correct copy of objects
  - is consistent with the physical times at which each operation occurred during execution

This is the strongest form of consistency.
Practical Linearizability

Consider the following scenario:

You (NY)  
Friend (CA)

Data Centers

write(a, x)  
write(a, y)  
read(a) → y

What are the challenges to linearizability?
Practical Linearizability

What are the challenges to linearizability?

You (NY)  \[ \text{write}(a, y) \]
Friend (CA)  \[ \text{write}(a, x) \]

What if:

- All clients send all operations to the CA data center.
- The CA data center propagates writes to the NY data center.
- No request returns until all propagation is finished.

Is this correct (does it display linearizability)?
**Practical Linearizability**

What are the challenges to linearizability?

You (NY) \(\rightarrow\) write(a, y) \(\rightarrow\) Friend (CA) write(a, x) \(\rightarrow\) read(a) \(\rightarrow\) y

What if:
- All clients **send all operations** to the CA data center.
- The CA data center **propagates writes** to the NY data center.
- No request returns **until all propagation is finished**.

Is this correct (does it display linearizability)?
- Yes
Practical Linearizability

What are the challenges to linearizability?

You (NY) \(\text{write}(a, y)\)
Friend (CA) \(\text{write}(a, x)\) \(\text{read}(a) \rightarrow y\)

What if:

- All clients send all operations to the CA data center.
- The CA data center propagates writes to the NY data center.
- No request returns until all propagation is finished.

Is this correct (does it display linearizability)?

- Yes

Is this performant?

- No
Practical Linearizability

What are the challenges to linearizability?

You (NY) \[\text{write}(a, x)\] Friend (CA) \[\text{write}(a, y)\] \[\text{read}(a) \rightarrow y\]

What if:
- All clients **send all operations** to the CA data center.
- The CA data center **propagates writes** to the NY data center.
- No request returns until all propagation is finished.

Is this correct (does it display linearizability)?
- Yes

Is this performant?
- No
Implementing Linearizability

Latency is very important!
- Amazon: 100 ms of added latency costs 1% in sales.
- Google: 500 extra ms in search page generation time dropped traffic by 20%.

Linearizability typically requires complete synchronization of replicas before a write operation returns.
- Read on any replica returns the most recent write
- This means writes must be synchronous

This means this approach is too expensive in a global setting.
- Cross-country RTT is ~20 ms minimum.
- Cross-oceanic is even longer!

It might still make sense for local replicas.
Passive (Primary-Backup) Replication

- **Request**: Requests are issued to the primary replica manager (RM), with unique ID.
- **Coordination**: The primary RM takes requests atomically, in order. Duplicate requests are detected by ID.
- **Execution**: The primary RM executes the request and stores its response.
- If the request is an **update**, it sends updates to all backup RMs (with 1-phase commit).
- **Response**: The primary RM sends its response to the client.
Chain Replication

Performance can be improved via chain replication [1].
- All writes go to the head of a chain of replicas.
- All reads go to the tail of the chain.

Is this linearizable?
Chain Replication

Performance can be improved via chain replication [1].

- All writes go to the had of a chain of replicas.
- All reads go to the tail of the chain.

Is this linearizable?

- It’s straightforward for non-overlapping operations
- What about overlapping?
Chain replication with Overlap

How are operations linearized if they overlap?

- The absolute order of overlapping operations can be implementation dependent.
- Consider a write that has propagated to N0 or N1 when a read arrives at N2.
How are operations linearized if they overlap?

- The absolute order of overlapping operations can be implementation dependent.
- Consider a write that has propagated to N0 or N1 when a read arrives at N2.
  - The imposed ordering is read-write.
How are operations linearized if they overlap?

- The absolute order of overlapping operations can be implementation dependent.

- Consider a write that has propagated to N0 or N1 when a read arrives at N2.
  - The imposed ordering is read-write.

- Consider a write that has propagated to N2 when a read arrives at N2.
Chain replication with Overlap

How are operations linearized if they overlap?

- The absolute order of overlapping operations can be implementation dependent.
- Consider a write that has propagated to N0 or N1 when a read arrives at N2.
  - The imposed ordering is read-write.
- Consider a write that has propagated to N2 when a read arrives at N2.
  - The imposed ordering is write-read.

Once a write becomes visible, all future reads will see it.
References

Required Readings

Optional Readings
Acknowledgements

These slides are based on slides from Steve Ko, used with permission.

Those slides contain the following attribution: These slides complain material developed and copyrighted by Indranil Gupta (UIUC).