CS 91: Cloud Systems & Datacenter Networks

Consistency & Event Ordering

Subtitle: One of the reasons why all the papers we read claim that building distributed systems is hard...
Consistency Model

• Contract between system and programmer/software

• Defines what is allowed to happen as multiple read and write operations happen in parallel

Node A
Node B
w(x)4
r(x)?

(Assume x is 0 at the beginning)
Cloud Computing?

• Paper talks about shared memory
  – On one machine, this is clearly a hard problem
  – Even harder in a distributed setting with failures

• Applicable to all kinds of distributed systems
  – If there are “multiple copies” of something in system
  – If multiple machines are accessing shared data

• Examples:
  – Facebook has multiple data centers
  – Many people editing a Google document
Consistency Models

- Lots of options out there.
- None is “right” or “wrong”.
- Fundamentally a trade-off:
  - Ease of programming (intuitive)
  vs.
  - Performance
“Simple” Distributed Example

These two machines are running programs that access the shared data.

Shared data. Let’s say it’s your Facebook status message.
“Simple” Distributed Example

Which value should the left node see?

(Options: A, B, “It depends”)
Intuition

• “It depends on when the events happened”

• Let’s suppose the right node initiated the write first. (It hit the write line of code first.)
Care to change your answer?

Which value should the left node see?

(Options: A, B, “It depends”)
Care to change your answer?

What actually happens here will depend on your consistency model.

If you said “B”: that’s impossible to guarantee!
Strict (Wall Clock) Consistency

• In practice, there is a delay for messages to go through network.

• Even if their two clocks are very closely synchronized, it will never be perfect.
Strict (Wall Clock) Consistency

We know that the write came first because I said so.

Nodes can’t always be sure!
Observation

• It probably doesn’t matter what time (wall clock) an event happened if everyone can agree on a sequence of events.
  – Forget about absolute time
  – Only care about relative order of events

• System gets to decide the order, not clock

• Basis for sequential consistency!
Sequential Consistency

• Still pretty strict and easy to reason about.

• “The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.” -Lamport (1979)
Sequential Consistency

- A single machine is always self-consistent.

- “The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.” - Lamport (1979)
Sequential Consistency

• A single machine is always self-consistent. (We’re going to assume this always.)
Sequential Consistency

• There must be some sequential order.

• “The result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.” - Lamport (1979)
Sequential Consistency

- There must be some sequential order.

- No requirement about how that order is determined, as long as every node agrees on the same order.

- In other words, if no node can prove the agreed upon order is wrong, it’s ok.
Sequential Consistency

What are the possible results for the reads now? (Hint: multiple options)

Left Node

Right Node

Read data

Write data (B)

r(□)?

r(□)?

w(□)B
Sequential Consistency

Left Node

Right Node

Sequence:
write, first read, second read

Sequence:
first read, write, second read
Sequential Consistency

Left Node

Right Node

Sequence: first read, second read, write

Left Node

Right Node

Sequence: ?

LeF Node

Right Node

Sequence: first read, second read, write
Implementing Sequential Consistency

• Typically easy if there is a single copy of data.
  – Let the node storing the data decide on the order

All requests for go to this server, it can determine order!

Total order:
Implementing Sequential Consistency

- Typically easy if there is a single copy of data.
  - Let the node storing the data decide on the order

All requests for \( A \) go to this server, it can determine order!

Total order: \[ \text{[ ] [ ] [ ]} \]
Implementing Sequential Consistency

• Typically easy if there is a single copy of data.
  – Let the node storing the data decide on the order

All requests for go to this server, it can determine order!

Total order:
Implementing Sequential Consistency

• Typically easy if there is a single copy of data.
  – Let the node storing the data decide on the order

All requests for \( A \) go to this server, it can determine order!

Total order:

Alternate total order:
Implementing Sequential Consistency

• Typically easy if there is a single copy of data.
  – Let the node storing the data decide on the order

Problem: can’t always have a single copy of shared data.

Maybe you want multiple copies for redundancy.
Maybe you want multiple copies that are closer to where they’ll be used (locality).
Two-Copy Example

Both nodes have a copy of the values X and Y.
Two-Copy Example

Inform one another if a value is written.
Two-Copy Example

Both nodes begin executing.

\[ A_0 : \text{Set } X = 1 \]
\[ A_1 : \text{Read } Y \]

\[ B_0 : \text{Set } Y = 1 \]
\[ B_1 : \text{Read } X \]

Using sequential consistency, three possible outcomes:

- A reads \( y = 1 \), B reads \( x = 1 \)
- A reads \( y = 0 \), B reads \( x = 1 \)
- A reads \( y = 1 \), B reads \( x = 0 \)
Two-Copy Example

Both nodes begin executing.

A₀: Set X = 1
A₁: Read Y

B₀: Set Y = 1
B₁: Read X

Event sequences:

A₀, B₀, A₁, B₁
A₀, B₀, B₁, A₁
B₀, A₀, A₁, B₁
B₀, A₀, B₁, A₁

Using sequential consistency, three possible outcomes:

A reads y=1,  B reads x=1
A reads y=0,  B reads x=1
A reads y=1,  B reads x=0
Two-Copy Example

Both nodes begin executing.

A₀: Set X = 1
A₁: Read Y

B₀: Set Y = 1
B₁: Read X

Event sequences:
A₀, A₁, B₀, B₁

Using sequential consistency, three possible outcomes:
A reads y=1,  B reads x=1
A reads y=0,  B reads x=1
A reads y=1,  B reads x=0
Two-Copy Example

Both nodes begin executing.

\[
\begin{align*}
A_0 &: \text{Set } X = 1 \\
A_1 &: \text{Read } Y \\
B_0 &: \text{Set } Y = 1 \\
B_1 &: \text{Read } X
\end{align*}
\]

Event sequences:

\[B_0, B_1, A_0, A_1\]

Using sequential consistency, three possible outcomes:

\[
\begin{align*}
\text{A reads } y=1, \quad \text{B reads } x=1 \\
\text{A reads } y=0, \quad \text{B reads } x=1 \\
\underline{\text{A reads } y=1, \quad \text{B reads } x=0}
\end{align*}
\]
Sequential Performance

\[
\begin{align*}
& A_0, B_0, A_1, B_1 \\
& A_0, B_0, B_1, A_1 \\
& B_0, A_0, A_1, B_1 \\
& B_0, A_0, B_1, A_1
\end{align*}
\]

\[
\begin{align*}
& B_0, B_1, A_0, A_1 \\
& A_0, A_1, B_0, B_1
\end{align*}
\]

Cannot happen:
A reads y=0, B reads x=0

• Regardless of which sequence, MUST do communication on reads, writes, or both, making them SLOW!
  – Access to memory: \(~100\) ns
  – Message send across local network: \(~1\) ms

Factor of 10,000 difference.
Two-Copy Example

Using sequential consistency, three possible outcomes:

A reads y=1,  B reads x=1
A reads y=0,  B reads x=1
A reads y=1,  B reads x=0
Not possible:
A reads y=0,  B reads x=0

In practice, would A and B both reading 0 be a bad thing?

It depends on the guarantees your system wants to provide…
Weaker Models

• Maybe: it’s ok if there is no total ordering of events, *unless the user tells you otherwise*

• Maybe: it’s ok if there is no total ordering *as long as a causal ordering is preserved*

• Maybe: it’s ok if there is no total ordering *as long as writes are eventually propagated*
Release Consistency

• Put the burden on the programmer to specify when it’s ok to be fast vs. slow & consistent.

• Programmer puts acquire() and release() around shared accesses.

• If A does acquire(X) ... release(X) and then B does acquire(X), it will see all of A’s writes.
Causal Consistency

• It’s ok for events to appear inconsistent if they are unrelated. Must be consistent if one event may have caused another.

• Keep track of the latest thing that each node has seen with a vector. Before observing a write, must observe all events that writer saw too.

• Provides a partial order on events rather than a total ordering.
• Suppose we have the following scenario:
  – Sally posts to Facebook, “Billy is missing!”
  – (Billy is at a friend’s house, sees message, calls mom)
  – Sally posts new message, “False alarm, he’s fine”
  – Sally’s friend James posts, “What a relief!”

• NOT causally consistent:
  – Third user, Henry, sees only:
Causal Consistency Example
(https://www.cs.berkeley.edu/~alig/papers/bolt-on-causal-consistency.pdf)

• Suppose we have the following scenario:
  – Sally posts to Facebook, “Billy is missing!”
  – (Billy is at a friend’s house, sees message, calls mom)
  – Sally posts new message, “False alarm, he’s fine”
  – Sally’s friend James posts, “What a relief!”

• NOT causally consistent:
  – Third user, Henry, sees only:
Causal Consistency Example
(https://www.cs.berkeley.edu/~alig/papers/bolt-on-causal-consistency.pdf)

• Suppose we have the following scenario:
  – Sally posts to Facebook, “Billy is missing!”
  – (Billy is at a friend’s house, sees message, calls mom)
  – Sally posts new message, “False alarm, he’s fine”
  – Sally’s friend James posts, “What a relief!”

• Causally consistent version:
  – Third user, Henry, sees only:
    Because James had seen Sally’s second post (which caused his response), Henry must also see it prior to seeing James’s.
Eventual Consistency

• Writes will propagate through system eventually. Stored values might diverge...

• (Think multiple git branches.)

• Advantage: very few messages required. Can work even if serious failures (network partitioned in two)

• Disadvantage: merging diverged changes can sometimes be hard (require human intervention)
Upcoming Paper Preview

• “Time, Clocks, and the Ordering of Events in a Distributed System”
  – Introduced notion of causal ordering

• “Reliable Communication in the Presence of Failures”
  – Dense paper, solutions to provide multiple forms of event ordering, taking failures into account

• Many more when we talk about replication...
Other Interesting Reads

  – Facebook’s real-life consistency problems (few years ago)

• [http://queue.acm.org/detail.cfm?id=2610533](http://queue.acm.org/detail.cfm?id=2610533)
  – Lots of good wide-area event ordering examples

• [http://www.linuxjournal.com/article/8211?page=0,0](http://www.linuxjournal.com/article/8211?page=0,0)
  – Discussion of how Linux handles many different processor architectures, each of which has its own consistency model