Distributed Systems II: More Linearizability

Where we are

- Consistency models
  \( \Rightarrow \) What order of operations can be observed on a concurrent data structure
  (given by an AOT with a spec. spec.)

- History: \( \langle \text{inv} \ldots \rangle \langle \text{req} \ldots \rangle \)

- Linearizability:
  An implementation is linearizable

\( \iff \)
any history \( H \) produced by impl.
can be extended to \( H' \) s.t.
we can find a sequential history \( S \) where \( \langle \text{complete}(H') \rangle \leq \langle S \rangle \)

- Linearizability - B

\( \langle H \rangle \leq \langle S \rangle \)
- **Locality**

  \[ H \text{ linearizable} \iff H/0 \text{ linearizable} \]
  
  (on other consistency that provides locality)
  
  for all objects \( o \) appearing in \( H \)

- **Non-blocking**

  \[ H \text{ linearizable} \iff H \text{ contains } \langle \text{inv} \circ o \circ \text{op} \rangle \text{ w/o } \langle \text{res} \circ o \circ v \rangle \Rightarrow \text{ we can find } v \text{ s.t. } H \circ \langle \text{res} \circ o \circ v \rangle \text{ is linearizable.} \]

- **Linearizable-B is not local & not non-blocking**

  \[ \Rightarrow \text{ saw two counter examples last class} \]

\[ H \]

\[ \begin{array}{cc}
I_0.\text{req}(x) & I_0.\text{em}(y) \\
R_y & \\
I_0.\text{del}(s) & I_0.\text{em}(s) \\
\end{array} \]

\[ H/0_1: \text{Lin-B} \]

\[ H/0_2: \text{Lin-B} \]

\[ H: \text{Lin-Bx} \]

\[ \text{Not} \]

\[ \text{Local} \]
Today
- A couple of other consistency models
  - Seq. Cons.
  - Strict Serializability
  - Comparing the strength of consistency models.
- Building linearizable "things"
- CAP
Sequential Consistency/Seq-Cst

- Often the strongest memory model supported by language specs: C++ std::memory-order::seq-cst
  Rust Ordering::SeqCst

- Guarantees
  - Can find a total order for all processes
  - Total order does not violate process order for any process.

Translating to our terminology

\[ H: \text{History}. \quad [\text{But } <_H \text{ is unused}] \]

\[ \text{For any process } p \in \mathcal{P}, \quad \leq_p : \text{order of operations.} \]

\[ H_p \leq_p \leq_{H_p} \]

[Remember, we assumed processes are sequential]

\[ H \rightarrow S \quad \text{s.t.} \quad \leq_p \subseteq \leq_s \cup \{p \in \mathcal{P}\} \]
$q_0\cdot\text{deq}(\cdot)\quad\text{OK}\times$

$P_0$

$P_1$

$\text{Seq\ cst? }\checkmark$

$\text{Linearizable? }\times$

$\text{Seq\ cst? }\checkmark$

$\text{Linearizable? }\times$

$q_0\cdot\text{enq}(\cdot)\quad\text{OK}\cdot$

$q_1\cdot\text{enq}(\cdot)\quad\text{OK}$

$q_1\cdot\text{deq}(\cdot)\quad\text{y}$

$q_0\cdot\text{deq}(\cdot)\quad\text{OK}\times$

$q_1\cdot\text{enq}(\cdot)\quad\text{OK}$
Strict Serializability

- Transaction
  - Sequence of operations issued by one process
  - Operations might target different objects
  - Must be **Isolated**

- **Strict** serializable
  - **Linearizable**
    - s.t. in linearization S
    - Cannot find \( o_1, o_2, o_3 \)
    - s.t. \( \text{Tran}(o_1) = \text{Tran}(o_3) \neq \text{Tran}(o_2) \)
    - \( o_1 <_S o_2 \land o_2 <_S o_3 \land o_1 < S o_3 \)
Non-blocking?

Local?

Observe

Linearizable $\implies$ Seq Cons.

Why? Processes are sequential

$\implies$ For any history H, process P

$\prec_P \subseteq \prec_H \subseteq \prec_S$

Linearizability $\leq$ Strict Serializability
Notion of stronger/weaker consistency model

Implications for the readings?

R[FL]

Building Linearizable Systems

Why?

- The main reason we read the Healthy Wing paper.
- Arguments:
  - Linearizability is easier for programmers to reason about
    - When do effects become visible?

Two Ends of the Spectrum

- Assume you are given a linearizable impl. of an ADT

  - Queue
    - Register → get 2
      - set 5
      - get: Returns value of last
        - set (or 1 if no
          - set has been called)

Sequential

\[
\begin{align*}
\text{spec} & \quad \begin{cases}
get() & \rightarrow \top \quad \checkmark \\
gset(5) \ gset(6) \ gset(7) \ gget() & \rightarrow 6 \quad \checkmark \\
gset(5) \ gset(6) \ gset(7) \ gget() & \rightarrow 6 \times
\end{cases}
\end{align*}
\]

Types of systems

- Collection of independent lin. impl. of ADT

  For example, KV Store

  (No txns, only single key operations,
Observe: Sequential spec. does not impose any ordering requirement b/w keys. Just use a set of lin. register

Relation to locality

- Collection of dependent ADTs

  Counter
  - Init value $\phi$
  - Inc $\rightarrow$ increment value, return OK
  - Dec $\rightarrow$ decrement value, return OK

- Get $\rightarrow$ current value
  Maybe track value in a lin. register
init:
  val.set(0)
get:
  return val.get()
inc:
  c = val.get()
  val.set(c+1)
dec:
  c = val.get()
  val.set(c-1)

Does this work?

Can we fix?

Practice

- One common approach in distributed system
  - Maintain a totally ordered log of operations
- Apply operations in logical order

- **Why?**
  - Need to replicate data for F.O.T.
  - Need to ensure replicas agree on order even for concurrent ops
  
  [Failures cannot break abstraction]

**Critically analyzing RIF1**

- Assumptions
  - RPCs; runtime
  - System - Durable storage & Rel.md

- Client

- Importance of exactly once to linearizability?

Does exactly once delivery $\Rightarrow$ Linearizability?
Does linearizability $\Rightarrow$ exactly once delivery?

- What does RiFL Provide?

CAP:

- Why?
  - Need to replicate data for F.T.
  - Need to ensure replicas agree on order even for concurrent Ops

[Failures cannot break abstraction]
@ Can replicas agree on order of concurrent ops without communicating?

Problem: Remember $R_0$ cannot distinguish between
- $R_1$ has failed
- Messages from $R_1$ are partition delayed (indep.)

In the absence of messages from $R_1$, $R_0$ has two options
- Wait "Not available"
  Processing might take unbounded time
- Process, assuming $R_1$ has failed
Ro & R₁ might disagree on results.

But Ro, R₁ are impl. details

→ Violate linearizabilities

"Not consistent"