Linearizability

Jinyang Li
Consistency model

- Consistency model = meaning of operations in the face of concurrency and failure
  - A contract between system designers and application programmers
  - A set of rules dictating what allowed system behaviors are
Spectrum of Consistency Models

- Eventual Consistency
- Causal Consistency
- Release Consistency
- Sequential Consistency [Lamport]
- Linearizability [Herliny] (aka external Consistency)

Weak/relaxed

- Easier to achieve good performance

Strong

- Easier to program
Consistency models are abstract

• Independent from any implementation
• Why specifying an abstract consistency model?
  – Application programmers do not have to understand the system implementation
  – System developers can reason correctness of implementation & optimizations
    • add caching
    • add replication
    • shard data across machines etc.
Application programmers’ expectation

Thread-1 (running on client machine 1)
Put("x", 1);
Put("y", 1);

Thread-2 (running on client machine 2)
Get("Y")=?;
Get("X")=?;

• What are expected values for Gets?
  • Put(x,1), Put(y,1), Get(y)=1,Get(x)=1
  • Put(x,1),Get(y)=0,Put(y,1), Get(x)=1
  • ...
  • Get(y)=0,Get(x)=0,Put(x,1),Put(y,1)

– 6 potential interleavings:
– 3 potential readings: (x=0,y=0) or (x=1 y=1) or (x=1 y=0)
A strawman key-value store

- System implementation:
  - Clients send writes to both replicas in parallel, wait for the fastest reply
  - Clients read from either replica
Does strawman satisfies programmers’ expectation?

• Programmers’ expectation: 3 read values
  – (x=0, y=0), or (x=1, y=1), or (x=1, y=0)

• Is the expectation fulfilled w/ a single server replica?

• Is the expectation fulfilled w/ 2 replicas?
  – Is x=0, y=1 possible under strawman?
How to define a strong consistency model?

**Reality**
- Concurrent execution
- Replicated data

**Simple abstract mental model**
- “Sequential” behavior
- “One-copy” of data
How to define a strong consistency model?

- Reduce a concurrent history to an equivalent serial execution history
How to define a strong consistency model?

- Not all equivalent serial execution history "make sense"
What are allowed equivalent serial history?

• Certain orders in the original execution should be preserved in the equivalent history

• What are possibilities?
  1. Global op issue order
  2. Global op completion order
  3. per-thread issue (and completion) order
  4. global “completion-to-issue” order

If op1 finishes before op2 starts, then op1 → op2 in equivalent history
1. Global issue order
2. Global completion order
3. Per-thread issue (and completion) order
4. Global completion-to-issue order
Put(x, 1) Put(y, 1)

Get(y) = 1 Get(x) = 1

client-1

client-2

Put(x, 1) Put(y, 1)

Get(y) = 1 Get(x) = 1

1. Global issue order
✗ 2. Global completion order
✗ 3. Per-thread issue (and completion) order
✔ 4. Global completion-to-issue order
client-1

Put(x, 1)  

Put(y, 1)

client-2

Get(y) = ?  

Get(x) = ?

Get(y) = 0  

Get(x) = 0  

Put(x, 1)  

Put(y, 1)

✗ 1. Global issue order
✗ 2. Global completion order
✔ 3. Per-thread issue (and completion) order
✗ 4. Global completion-to-issue order
Put(x,1)  
Put(y,1)  

Get(y)=?  
Get(x)=?  

Get(x)=0  
Put(x,1)  
Put(y,1)  
Get(y)=1  

✗ 1. Global issue order  
✗ 2. Global completion order  
✗ 3. Per-thread issue (and completion) order  
✗ 4. Global completion-to-issue order
Strong consistency models

• Preserving global issue/completion order is impractical
  → extreme blocking behavior

Can not complete Put(y, 1) without waiting for Put(x, 1) to finish → block on arbitrary machine
Strong consistency models

• Sequential consistency
  – equivalent serial history must obey per-process issue/completion order

• Linearizability
  – equivalent serial history must obey global “completion-to-issue” order

• Which one is stronger?
1. Global issue order
2. Global completion order

stronger

4. Global completion-to-issue order (Linearizability)

stronger

3. Per-thread issue (and completion) order (sequential consistency)
Sequential consistency vs. linearizability

client-1
- Put(x, 1)
- Put(y, 1)

client-2
- Get(y) = ?
- Get(x) = ?

Get(y) = 0
- Get(x) = 0
- Put(x, 1)
- Put(y, 1)

sequentially consistent but not linearizable
Sequential consistency vs. linearizability

• Does the difference between the two matter?
  – Matters only when external communication between threads could be present

• Both are equally good w/o external communication
  – e.g. Multi-threaded programs sharing sequential consistent global memory
Why linearizability is a good strong model?

• It is strong
  – Strongest possible practical model
• It allows scalable system design
  – (the local property) if each object is linearizable, then the whole system is linearizable
  – Scale system → shard objects across servers
Recap exercise: which of the following histories are linearizable?

(a) client-1: \( \text{Put}(x,1) \) \( \text{Get}(x)=1 \) \( \text{Get}(x)=1 \)
   client-2: \( \text{Put}(x,2) \) \( \text{Get}(x)=2 \)

(b) client-1: \( \text{Put}(x,1) \) \( \text{Get}(x)=1 \) \( \text{Get}(x)=1 \)
   client-2: \( \text{Put}(x,2) \) \( \text{Get}(x)=2 \)

(c) client-1: \( \text{Put}(x,1) \) \( \text{Get}(x)=1 \) \( \text{Get}(x)=1 \)
   client-2: \( \text{Put}(x,2) \) \( \text{Get}(x)=2 \)

(d) client-1: \( \text{Put}(x,1) \)
   client-2: \( \text{Put}(x,2) \) \( \text{Get}(x)=2 \)
What are the implementations of linearizability

• Case-study: key-value store
  – Single server
  – N servers: server-i stores keys, k mod N = i
  – 3 servers replicating data using viewstamp replication

write(x,1)  read(x)  write(y,2)  write(x,3)  read(x)

VR log contains both read and write ops
How about this optimization

• Viewstamp with read optimization.
  – read at any arbitrary server
  – read at the primary
The “correct” VR read optimization

request lease from f+1 nodes (including self)

lease held till 18:25:30
lease held till 18:24:50

viewchange does not proceed because of valid lease
Strong consistency models in practice

• The memory model of Intel CPU?
• Popular key-value stores
  – S3
  – Dynamo
  – MySQL with asynchronous replication