Spanner

Storage insights

COS 518: *Advanced Computer Systems*
Lecture 6

Michael Freedman

2PL & OCC = strict serialization

- Provides semantics as if only one transaction was running on DB at time, in serial order

  + Real-time guarantees

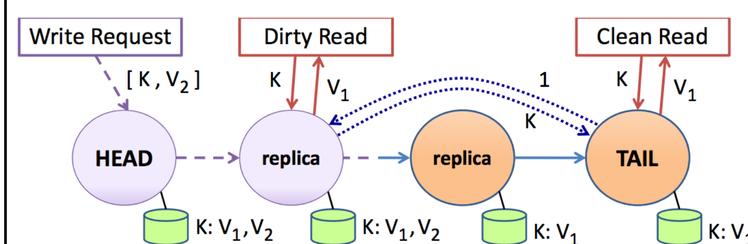
- 2PL: Pessimistically get all the locks first
- OCC: Optimistically create copies, but then recheck all read + written items before commit

Multi-version concurrency control

Generalize use of multiple versions of objects

Multi-version concurrency control

- Maintain multiple versions of objects, each with own timestamp. Allocate correct version to reads.
- Prior example of MVCC:



Multi-version concurrency control

- Maintain multiple versions of objects, each with own timestamp. Allocate correct version to reads.
- Unlike 2PL/OCC, reads never rejected
- Occasionally run garbage collection to clean up

5

MVCC Intuition

- Split transaction into read set and write set
  – All reads execute as if one "snapshot"
  – All writes execute as if one later "snapshot"
- Yields snapshot isolation < serializability

6

Serializability vs. Snapshot isolation

- Intuition: Bag of marbles: ½ white, ½ black
- Transactions:
  – T1: Change all white marbles to black marbles
  – T2: Change all black marbles to white marbles
- Serializability (2PL, OCC)
  – $T1 \rightarrow T2$ or $T2 \rightarrow T1$
  – In either case, bag is either ALL white or ALL black
- Snapshot isolation (MVCC)
  – $T1 \rightarrow T2$ or $T2 \rightarrow T1$ or $T1 \,||\, T2$
  – Bag is ALL white, ALL black, or ½ white ½ black

7

Timestamps in MVCC

- Transactions are assigned timestamps, which may get assigned to objects those txns read/write
- Every object version O_V has both read and write TS
  – ReadTS: Largest timestamp of txn that reads O_V
  – WriteTS: Timestamp of txn that wrote O_V

8

Executing transaction T in MVCC

- Find version of object O to read:
  - # Determine the last version written before read snapshot time
  - Find O_V s.t. max { WriteTS(O_V) | WriteTS(O_V) <= TS(T) }
  - ReadTS(O_V) = max(TS(T), ReadTS(O_V))
  - Return O_V to T
- Perform write of object O or abort if conflicting:
  - Find O_V s.t. max { WriteTS(O_V) | WriteTS(O_V) <= TS(T) }
  - # Abort if another T' exists and has read O after T
  - If ReadTS(O_V) > TS(T)
    - Abort and roll-back T
  - Else
    - Create new version O_W
    - Set ReadTS(O_W) = WriteTS(O_W) = TS(T)

9

Digging deeper

txn txn txn

TS = 3 **TS = 4** **TS = 5**

Notation

W(1) = 3: Write creates version 1 with WriteTS = 3

R(1) = 3: Read of version 1 returns timestamp 3

write(O)
by TS=3

O

10

Digging deeper

txn txn txn

TS = 3 **TS = 4** **TS = 5**

Notation

W(1) = 3: Write creates version 1 with WriteTS = 3

R(1) = 3: Read of version 1 returns timestamp 3

W(1) = 3
R(1) = 3

write(O)
by TS=5

O

11

Digging deeper

txn txn txn

TS = 3 **TS = 4** **TS = 5**

Notation

W(1) = 3: Write creates version 1 with WriteTS = 3

R(1) = 3: Read of version 1 returns timestamp 3

W(1) = 3
R(1) = 3

W(2) = 5
R(2) = 5

O

write(O)
by TS = 4

Find v such that max WriteTS(v) <= (TS = 4)
⇒ v = 1 has (WriteTS = 3) <= 4
If ReadTS(1) > 4, abort
⇒ 3 > 4: false
Otherwise, write object

12

Digging deeper

Notation

txn txn txn

TS = 3 **TS = 4** **TS = 5**

W(1) = 3: Write creates version 1 with WriteTS = 3

R(1) = 3: Read of version 1 returns timestamp 3

W(1) = 3 **W(3) = 4** **W(2) = 5**
R(1) = 3 **R(3) = 4** **R(2) = 5**

O ⟶

Find v such that max WriteTS(v) <= (TS = 4)
⇒ v = 1 has (WriteTS = 3) <= 4
If ReadTS(1) > 4, abort
⇒ 3 > 4: false
Otherwise, write object

13

Digging deeper

Notation

txn txn txn

TS = 3 **TS = 4** **TS = 5**

W(1) = 3: Write creates version 1 with WriteTS = 3

R(1) = 3: Read of version 1 returns timestamp 3

W(1) = 3
R(1) = 5

O ⟶

BEGIN Transaction
tmp = READ(O)
WRITE (O, tmp + 1)
END Transaction

Find v such that max WriteTS(v) <= (TS = 5)
⇒ v = 1 has (WriteTS = 3) <= 5
Set R(1) = max(5, R(1)) = 5

14

Digging deeper

Notation

txn txn txn

TS = 3 **TS = 4** **TS = 5**

W(1) = 3: Write creates version 1 with WriteTS = 3

R(1) = 3: Read of version 1 returns timestamp 3

W(1) = 3 **W(2) = 5**
R(1) = 5 **R(2) = 5**

O ⟶

BEGIN Transaction
tmp = READ(O)
WRITE (O, tmp + 1)
END Transaction

Find v such that max WriteTS(v) <= (TS = 5)
⇒ v = 1 has (WriteTS = 3) <= 5
If ReadTS(1) > 5, abort
⇒ 5 > 5: false
Otherwise, write object

15

Digging deeper

Notation

txn txn txn

TS = 3 **TS = 4** **TS = 5**

W(1) = 3: Write creates version 1 with WriteTS = 3

R(1) = 3: Read of version 1 returns timestamp 3

W(1) = 3 **W(2) = 5**
R(1) = 5 **R(2) = 5**

O ⟶

write(O)
by TS = 4

Find v such that max WriteTS(v) <= (TS = 4)
⇒ v = 1 has (WriteTS = 3) <= 4
If ReadTS(1) > 4, abort
⇒ 5 > 4: **true**

16

Digging deeper

txn TS = 3 | txn TS = 4 | txn TS = 5

Notation

W(1) = 3: Write creates version 1 with WriteTS = 3

R(1) = 3: Read of version 1 returns timestamp 3

W(1) = 3
R(1) = 5

W(2) = 5
R(2) = 5

O ⟶

BEGIN Transaction
tmp = READ(O)
WRITE (P, tmp + 1)
END Transaction

Find v such that max WriteTS(v) <= (TS = 4)
⇒ v = 1 has (WriteTS = 3) <= 4
Set R(1) = max(4, R(1)) = 5

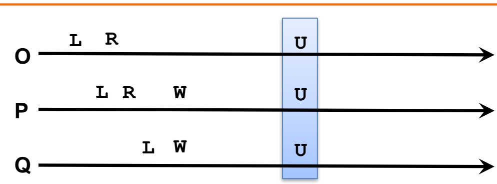
Then write on P succeeds as well

17

Distributed Transactions

18

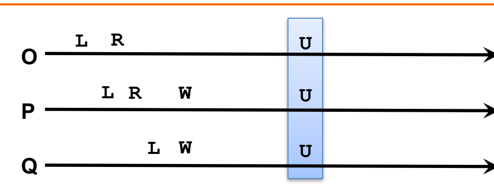
Consider partitioned data over servers



- Why not just use 2PL?
  - Grab locks over entire read and write set
  - Perform writes
  - Release locks (at commit time)

19

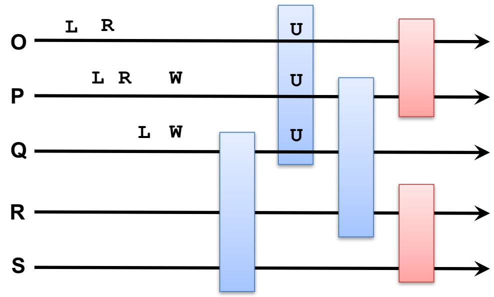
Consider partitioned data over servers



- How do you get serializability?
  - On single machine, single COMMIT op in the WAL
  - In distributed setting, assign global timestamp to txn (at sometime after lock acquisition and before commit)
    - Centralized txn manager
    - Distributed consensus on timestamp (not all ops)

20

Strawman: Consensus per txn group?



- Single Lamport clock, consensus per group?
  - Linearizability composes!
  - But doesn't solve concurrent, non-overlapping txn problem

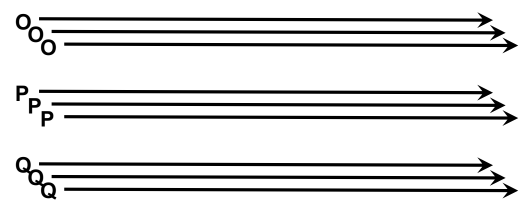
Spanner: Google's Globally-Distributed Database

OSDI 2012

Google's Setting

- Dozens of zones (datacenters)
- Per zone, 100-1000s of servers
- Per server, 100-1000 partitions (tablets)
- Every tablet replicated for fault-tolerance (e.g., 5x)

Scale-out vs. fault tolerance



- Every tablet replicated via Paxos (with leader election)
- So every "operation" within transactions across tablets actually a replicated operation within Paxos RSM
- Paxos groups can stretch across datacenters!
  - (COPS took same approach *within* datacenter)

Disruptive idea:

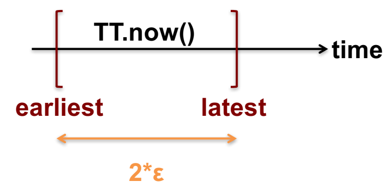
Do clocks **really** need to be arbitrarily unsynchronized?

Can you engineer some max divergence?

25

TrueTime

- "Global wall-clock time" with bounded uncertainty

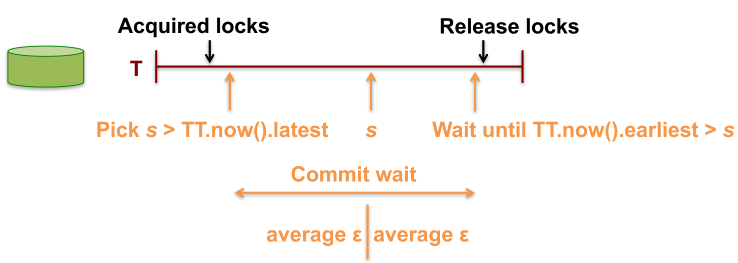


Consider event e_{now} which invoked tt = TT.new():

Guarantee: $tt.earliest <= t_{abs}(e_{now}) <= tt.latest$

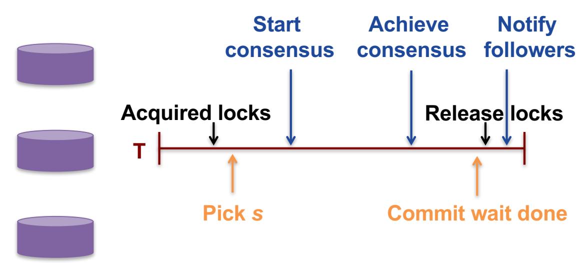
26

Timestamps and TrueTime



27

Commit Wait and Replication



28

Client-driven transactions

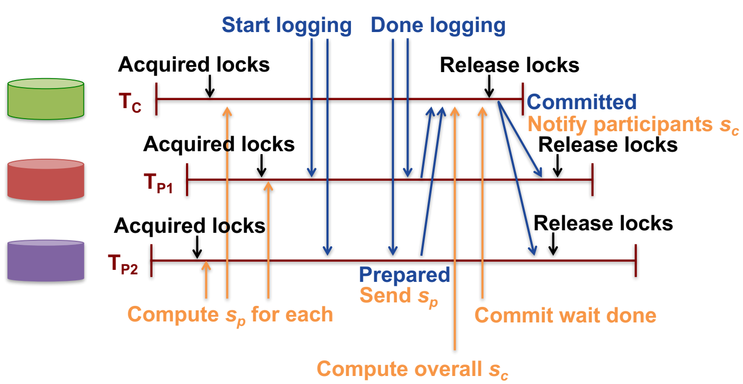
Client:

1. Issues reads to leader of each tablet group, which acquires read locks and returns most recent data
2. Locally performs writes
3. Chooses coordinator from set of leaders, initiates commit
4. Sends commit message to each leader, include identify of coordinator and buffered writes
5. Waits for commit from coordinator

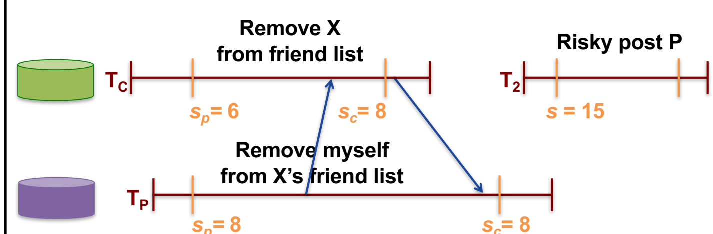
Commit Wait and 2-Phase Commit

- On commit msg from client, leaders acquire local write locks
  - If non-coordinator:
    - Choose prepare ts > previous local timestamps
    - Log prepare record through Paxos
    - Notify coordinator of prepare timestamp
  - If coordinator:
    - Wait until hear from other participants
    - Choose commit timestamp >= prepare ts, > local ts
    - Logs commit record through Paxos
    - Wait commit-wait period
    - Sends commit timestamp to replicas, other leaders, client
- All apply at commit timestamp and release locks

Commit Wait and 2-Phase Commit



Example



Time	<8	8	15
My friends	[X]	[]	
My posts			[P]
X's friends	[me]	[]	

Read-only optimizations

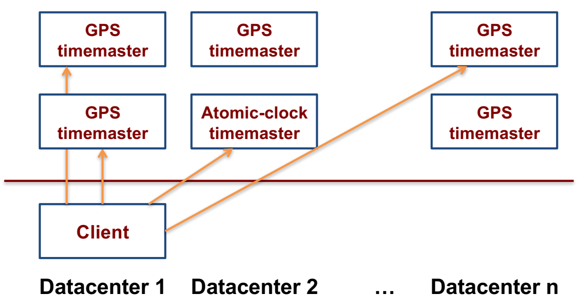
- Given global timestamp, can implement read-only transactions lock-free (snapshot isolation)
- Step 1: Choose timestamp s_{read} = TT.now.latest()
- Step 2: Snapshot read (at s_{read}) to each tablet
  – Can be served by any up-to-date replica

Disruptive idea:

Do clocks **really** need to be arbitrarily unsynchronized?

Can you engineer some max divergence?

TrueTime Architecture



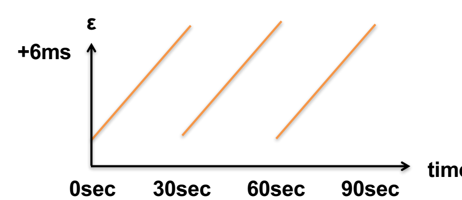
Compute reference [earliest, latest] = now ± ε

TrueTime implementation

now = reference now + local-clock offset

ε = reference ε + worst-case local-clock drift

= 1ms + 200 μs/sec



- What about faulty clocks?
  – Bad CPUs 6x more likely in 1 year of empirical data

Known unknowns > unknown unknowns

Rethink algorithms to reason about uncertainty

The case for log storage:
Hardware tech affecting software design

Latency Numbers Every Programmer Should Know

latency.txt June 7, 2012

```
Latency Comparison Numbers
--------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns
Compress 1K bytes with Zippy             3,000   ns        3 us
Send 1K bytes over 1 Gbps network       10,000   ns       10 us
Read 4K randomly from SSD*             150,000   ns      150 us
Read 1 MB sequentially from memory     250,000   ns      250 us
Round trip within same datacenter      500,000   ns      500 us
Read 1 MB sequentially from SSD*     1,000,000   ns    1,000 us    1 ms
Disk seek                           10,000,000   ns   10,000 us   10 ms
Read 1 MB sequentially from disk    20,000,000   ns   20,000 us   20 ms
Send packet CA->Netherlands->CA    150,000,000   ns  150,000 us  150 ms
```

From https://gist.github.com/jboner/2841832

See also https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html

~2016

Seagate ($50)
1TB HDD 7200RPM
Model: STD1000DM003-1SB10C

Operation	HDD Performance
Sequential Read	176 MB/s
Sequential Write	190 MB/s
Random Read 4KiB	0.495 MB/s 121 IOPS
Random Write 4KiB	0.919 MB/s 224 IOPS
DQ Random Read 4KiB	1.198 MB/s 292 IOPS
DQ Random Write 4KiB	0.929 MB/s 227 IOPS

http://www.tomshardware.com/answers/id-3201572/good-normal-read-write-speed-hdd.html

~2016

Seagate ($50)
1TB HDD 7200RPM
Model: STD1000DM003-1SB10C

Samsung ($330)
512 GB 960 Pro NVMe PCIe M.2
Model: MZ-V6P512BW

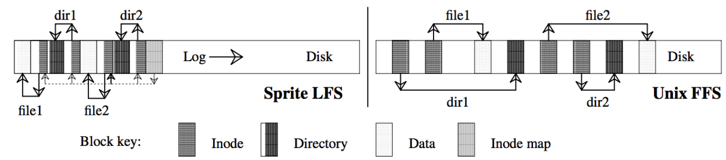
Operation	HDD Performance	SSD Performance
Sequential Read	176 MB/s	2268 MB/s
Sequential Write	190 MB/s	1696 MB/s
Random Read 4KiB	0.495 MB/s 121 IOPS	44.9 MB/s 10,962 IOPS
Random Write 4KiB	0.919 MB/s 224 IOPS	151 MB/s 36,865 IOPS
DQ Random Read 4KiB	1.198 MB/s 292 IOPS	348 MB/s 84961 IOPS
DQ Random Write 4KiB	0.929 MB/s 227 IOPS	399 MB/s 97,412 IOPS

http://www.tomshardware.com/answers/id-3201572/good-normal-read-write-speed-hdd.html
http://ssd.userbenchmark.com/SpeedTest/182182/Samsung-SSD-960-PRO-512GB

The Design and Implementation of a Log-Structured File System

Mendel Rosenblum and John K. Ousterhout

- **Idea:** Traditionally disks laid out with spatial locality due to cost of seeks
- **Observation:** main memory getting bigger → most reads from memory
- **Implication:** Disk workloads now write-heavy → avoid seeks → write log
- **New problem:** Many seeks to read, need to occasionally defragment
- **New tech solution:** SSDs → seeks cheap, erase blocks change defrag



This paper will appear in the *Proceedings of the 13th ACM Symposium on Operating Systems Principles* and the February 1992 *ACM Transactions on Computer Systems.*