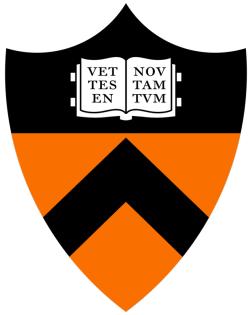
Databases



COS 418/518: Distributed Systems

Lecture 19

Jialin Ding, Mike Freedman

Rest of the semester

- Lectures
  - Databases
  - Blockchains
  - AI systems
  - Reasoning about performance
- Assignments
  - Assignment 4 due Tuesday, November 25
  - Assignment 5 due Friday, December 12 (no late days)
- Final: Wednesday, December 17
  - Only covers material from the second half of the semester
  - More logistical details to come (will post to Ed)

Today

- Background on databases
- More on isolation
  - MVCC and snapshot isolation
- More on sharding
  - Data distribution strategies

Key-value stores vs. relational databases

- Key-value stores
  - Get(K)
  - Put(K, V)
  - Delete(K)

Key	Value
as1234	{“name”: “Alice Smith”, “major”: “COS”}
bw5678	{“name”: “Bob Williams”, “major”: “ECE”}

Key-value stores vs. relational databases

- Key-value stores
  - Get(K)
  - Put(K, V)
  - Delete(K)
- Relational databases
  - Tables (i.e., “relations”) store records
  - Can select/filter records by different columns
  - Tables can be joined
  - Fixed schema

Key	Value
as1234	{“name”: “Alice Smith”, “major”: “COS”}
bw5678	{“name”: “Bob Williams”, “major”: “ECE”}

netid	first_name	last_name	major
as1234	Alice	Smith	COS
bw5678	Bob	Williams	ECE

major	enrollment
COS	400
ECE	300

Today

- Background on databases
- More on isolation
  - MVCC and snapshot isolation
- More on sharding
  - Data distribution strategies

MVCC: Multi-version concurrency control

- Store multiple versions of each record

netid	first_name	last_name	major	version
as1234	Alice	Smith	COS	0
bw5678	Bob	Williams	ECE	0

MVCC: Multi-version concurrency control

- Store multiple versions of each record

netid	first_name	last_name	major	version
as1234	Alice	Smith	COS	0
as1234	Alice	Smith	ECO	1
bw5678	Bob	Williams	ECE	0

MVCC: Multi-version concurrency control

- Store multiple versions of each record
  - Allows “time travel”
  - Enables snapshot isolation

netid	first_name	last_name	major	start_ts	end_ts
as1234	Alice	Smith	COS	Sep 2024	Oct 2025
as1234	Alice	Smith	ECO	Oct 2025	present
bw5678	Bob	Williams	ECE	Sep 2025	present

MVCC: Multi-version concurrency control

- Store multiple versions of each record
  - Allows “time travel”
  - Enables snapshot isolation

netid	first_name	last_name	major	start_xid	end_xid
as1234	Alice	Smith	COS	100	200
as1234	Alice	Smith	ECO	200	inf
bw5678	Bob	Williams	ECE	150	inf

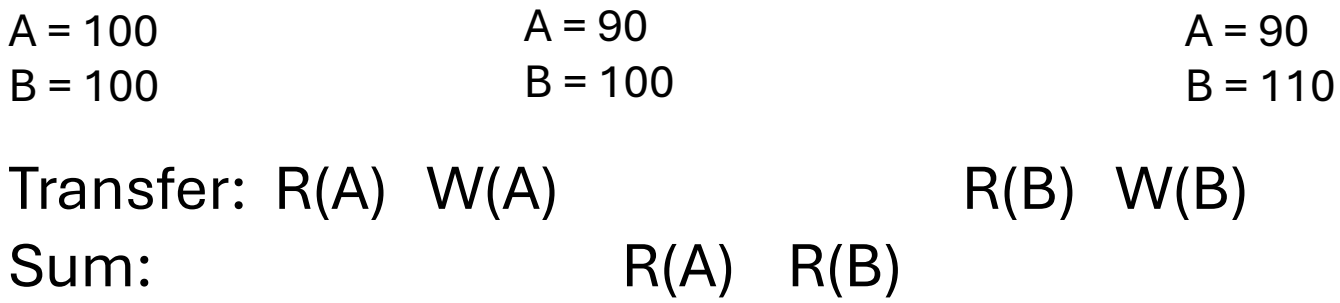
Snapshot Isolation

- A transaction reads from a snapshot taken at start time
- No locks for reads!
- Conflicts may arise due to writes
  - If two transactions write the same record, the first transaction commits and the second transaction aborts (write-write conflict)
  - Does **not** check for read-write conflicts

T1: R(A)->10 R(A)->20
T2: W(A=20)

Snapshot Isolation

- A transaction reads from a snapshot taken at start time
- No locks for reads!
- Conflicts may arise due to writes
  - If two transactions write the same record, the first transaction commits and the second transaction aborts (write-write conflict)
  - Does **not** check for read-write conflicts



Snapshot Isolation in Postgres

- Each transaction is committed with a monotonically increasing transaction ID (xid)
  - New records created by transaction will have xmin = xid
  - Records deleted by transaction will have xmax = xid

Snapshot Isolation in Postgres

key	value	xmin	xmax
A	10	0	inf

T1 (xid=1)

W(A, 20)

Snapshot Isolation in Postgres

key	value	xmin	xmax
A	10	0	1
A	20	1	inf

T1 (xid=1)

W(A, 20)

Snapshot Isolation in Postgres

- Each transaction is committed with a monotonically increasing transaction ID (xid)
  - New records created by transaction will have xmin = xid
  - Records deleted by transaction will have xmax = xid
- Snapshot is defined by three variables
  - xmin: oldest in-progress transaction ID at the time the snapshot was taken
  - xmax: next available transaction ID (not taken by committed or in-progress transactions)
  - xip: list of concurrent in-progress transaction IDs

Snapshot Isolation in Postgres

nextXID = 1

key	value	xmin	xmax
A	10	0	inf

T1

xmin = 1
xmax = 1
xip = {}

Snapshot Isolation in Postgres

- Each transaction is committed with a monotonically increasing transaction ID (xid)
  - New records created by transaction will have xmin = xid
  - Records deleted by transaction will have xmax = xid
- Snapshot is defined by three variables
  - xmin: oldest in-progress transaction ID at the time the snapshot was taken
  - xmax: next available transaction ID (not taken by committed or in-progress transactions)
  - xip: list of concurrent in-progress transaction IDs
- Visibility check for a given record
  - Record must be created
    - xmin < snapshot xmin, OR
    - Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip
  - Record must not be deleted
    - xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation in Postgres

nextXID = 1

key	value	xmin	xmax
A	10	0	inf

T1

xmin = 1
xmax = 1
xip = {}

R(A) -> ?

Record must be created

- xmin < snapshot xmin, OR
- Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip

Record must not be deleted

- xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation in Postgres

nextXID = 1

key	value	xmin	xmax
A	10	0	inf

T1

xmin = 1
xmax = 1
xip = {}

R(A) -> 10

Record must be created
- xmin < snapshot xmin, OR
- Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip

Record must not be deleted
- xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation in Postgres

nextXID = 2

key	value	xmin	xmax
A	10	0	inf

T1 (xid=1)

xmin = 1
xmax = 1
xip = {}

R(A) -> 10
W(A, 20)

Record must be created

- xmin < snapshot xmin, OR
- Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip

Record must not be deleted

- xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation in Postgres

nextXID = 2

key	value	xmin	xmax
A	10	0	1
A	20	1	inf

T1 (xid=1)

xmin = 1
xmax = 1
xip = {}

R(A) -> 10
W(A, 20)

Record must be created

- xmin < snapshot xmin, OR
- Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip

Record must not be deleted

- xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation in Postgres

nextXID = 2

key	value	xmin	xmax
A	10	0	1
A	20	1	inf

T1 (xid=1)	T2
xmin = 1 xmax = 1 xip = {}	xmin = 1 xmax = 2 xip = {1}
R(A) -> 10 W(A, 20)	
	R(A) -> ?

Record must be created

- xmin < snapshot xmin, OR
- Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip

Record must not be deleted

- xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation in Postgres

nextXID = 2

key	value	xmin	xmax
A	10	0	1
A	20	1	inf

T1 (xid=1)	T2
xmin = 1 xmax = 1 xip = {}	xmin = 1 xmax = 2 xip = {1}
R(A) -> 10 W(A, 20)	
	R(A) -> 10

Record must be created

- xmin < snapshot xmin, OR
- Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip

Record must not be deleted

- xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation in Postgres

nextXID = 3

key	value	xmin	xmax
A	10	0	1
A	20	1	inf
B	30	2	inf

T1 (xid=1)

xmin = 1
xmax = 1
xip = {}

R(A) -> 10
W(A, 20)

T2 (xid=2)

xmin = 1
xmax = 2
xip = {1}

R(A) -> 10
W(B, 30)
commit

Record must be created

- xmin < snapshot xmin, OR
- Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip

Record must not be deleted

- xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation in Postgres

nextXID = 3

key	value	xmin	xmax
A	10	0	1
A	20	1	inf
B	30	2	inf

T1 (xid=1)

xmin = 1
xmax = 1
xip = {}

R(A) -> 10
W(A, 20)

T3

xmin = 1
xmax = 3
xip = {1}

D(B)

Record must be created

- xmin < snapshot xmin, OR
- Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip

Record must not be deleted

- xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation in Postgres

nextXID = 4

key	value	xmin	xmax
A	10	0	1
A	20	1	inf
B	30	2	3

T1 (xid=1)

xmin = 1
xmax = 1
xip = {}

R(A) -> 10
W(A, 20)

T3 (xid=3)

xmin = 1
xmax = 3
xip = {1}

D(B)

Record must be created

- xmin < snapshot xmin, OR
- Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip

Record must not be deleted

- xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation in Postgres

nextXID = 4

key	value	xmin	xmax
A	10	0	1
A	20	1	inf
B	30	2	3

T1 (xid=1)

xmin = 1
xmax = 1
xip = {}

R(A) -> 10
W(A, 20)

T3 (xid=3)

xmin = 1
xmax = 3
xip = {1}

D(B)
R(A) -> 10

Record must be created

- xmin < snapshot xmin, OR
- Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip

Record must not be deleted

- xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation in Postgres

nextXID = 4

key	value	xmin	xmax
A	10	0	1
A	20	1	inf
B	30	2	3

T1 (xid=1)

xmin = 1
xmax = 1
xip = {}

R(A) -> 10
W(A, 20)

T3 (xid=3)

xmin = 1
xmax = 3
xip = {1}

D(B)
R(A) -> 10
W(A, 40)

Record must be created

- xmin < snapshot xmin, OR
- Snapshot xmin <= xmin < snapshot xmax AND xmin not in xip

Record must not be deleted

- xmax >= snapshot xmax OR xmax in xip

Snapshot Isolation vs. Serializability

- Serializability is a stronger isolation level
  - Snapshot isolation allows schedules that are not serializable (next slide)
- Snapshot isolation is more scalable
  - Reads and writes do not block each other, unlike 2PL

Write skew

- Occurs when transactions **read overlapping sets of data** but **write disjoint sets**
  - Snapshot isolation only checks for write-write conflicts on the same record

key	value
A	1
B	0

T1 (turn all 1s to 0s)	T2 (turn all 0s to 1s)
R(A)	R(A)
R(B)	R(B)
W(A, 0)	W(B, 1)

Write skew

- Occurs when transactions **read overlapping sets of data** but **write disjoint sets**
  - Snapshot isolation only checks for write-write conflicts on the same record

key	value
A	1
B	0

T1 (turn all 1s to 0s)	T2 (turn all 0s to 1s)
R(A)	R(A)
R(B)	R(B)
W(A, 0)	W(B, 1)

key	value
A	0
B	1

Not serializable!

Isolation vs. consistency

- Isolation is about whether concurrent transactions interfere with each other
  - Isolation models: serializable, snapshot isolation
- Consistency is about whether nodes see the same state
  - Consistency models: linearizable, causal+, eventual

Today

- Background on databases
- More on isolation
  - MVCC and snapshot isolation
- More on sharding
  - Data distribution strategies

Data distribution strategies

- Words that mean similar things
  - Sharding
  - Partitioning
    - Careful: could refer to data partitioning or network partitioning
  - Distribution
    - Careful: data can be distributed without being sharded

Data distribution strategies

- Example: distribute data across three nodes

	Node 1	Node 2	Node 3
Range	Netid in [aa0000, hz9999]	Netid in [ia0000, sz0000]	Netid in [ta0000, zz9999]
Round robin			
Hash			
All/broadcast			

Data distribution strategies

- Example: distribute data across three nodes

	Node 1	Node 2	Node 3
Range	Netid in [aa0000, hz9999]	Netid in [ia0000, sz0000]	Netid in [ta0000, zz9999]
Round robin	Row 1, row 4, row 7, …	Row 2, row 5, row 8, …	Row 3, row 6, row 9, …
Hash			
All/broadcast			

Data distribution strategies

- Example: distribute data across three nodes

	Node 1	Node 2	Node 3
Range	Netid in [aa0000, hz9999]	Netid in [ia0000, sz0000]	Netid in [ta0000, zz9999]
Round robin	Row 1, row 4, row 7, …	Row 2, row 5, row 8, …	Row 3, row 6, row 9, …
Hash	Hash(netid) mod 3 = 0	Hash(netid) mod 3 = 1	Hash(netid) mod 3 = 2
All/broadcast			

Data distribution strategies

- Example: distribute data across three nodes

	Node 1	Node 2	Node 3
Range	Netid in [aa0000, hz9999]	Netid in [ia0000, sz0000]	Netid in [ta0000, zz9999]
Round robin	Row 1, row 4, row 7, …	Row 2, row 5, row 8, …	Row 3, row 6, row 9, …
Hash	Hash(netid) mod 3 = 0	Hash(netid) mod 3 = 1	Hash(netid) mod 3 = 2
All/broadcast	All	All	All

Data distribution strategies

	Range	Round Robin	Hash	All/broadcast
Need to select a column?	Yes	No	Yes	No
Load balancing	Hard	Easy	Easy	N/A

Data distribution strategies

	Range	Round Robin	Hash	All/broadcast
Need to select a column?	Yes	No	Yes	No
Load balancing	Hard	Easy	Easy	N/A
Impact on joins	Can be useful but makes load balancing worse	Useless	Good for large tables	Good for small tables

Distribution strategies and joins

netid	first_name	last_name	major
as1234	Alice	Smith	COS
bw5678	Bob	Williams	ECE

major	enrollment
ECE	300
COS	400

Round robin distribution

Joined records are not co-located on the same node
-> network communication overhead

Node 1

netid	first_name	last_name	major
as1234	Alice	Smith	COS

major	enrollment
ECE	300

Node 2

netid	first_name	last_name	major
bw5678	Bob	Williams	ECE

major	enrollment
COS	400

Distribution strategies and joins

netid	first_name	last_name	major
as1234	Alice	Smith	COS
bw5678	Bob	Williams	ECE

major	enrollment
ECE	300
COS	400

Hash distribution (on “major” column for both tables)

Joined records are co-located on the same node
-> No network communication overhead

Node 1

netid	first_name	last_name	major
as1234	Alice	Smith	COS

major	enrollment
COS	400

Node 2

netid	first_name	last_name	major
bw5678	Bob	Williams	ECE

major	enrollment
ECE	300

Distribution strategies and joins

netid	first_name	last_name	major	grad_yr
as1234	Alice	Smith	COS	2027
bw5678	Bob	Williams	ECE	2026

major	enrollment
ECE	300
COS	400

grad_yr	enrollment
2026	1500
2027	1600

Node 1

netid	first_name	last_name	major	grad_yr
as1234	Alice	Smith	COS	2027

major	enrollment
COS	400

Node 2

netid	first_name	last_name	major	grad_yr
bw5678	Bob	Williams	ECE	2026

major	enrollment
ECE	300

Distribution strategies and joins

netid	first_name	last_name	major	grad_yr
as1234	Alice	Smith	COS	2027
bw5678	Bob	Williams	ECE	2026

major	enrollment
ECE	300
COS	400

grad_yr	enrollment
2026	1500
2027	1600

All/broadcast distribution for small tables

Data is duplicated but network communication is minimized

Node 1

netid	first_name	last_name	major	grad_yr
as1234	Alice	Smith	COS	2027

major	enrollment
COS	400

grad_yr	enrollment
2026	1500
2027	1600

Node 2

netid	first_name	last_name	major	grad_yr
bw5678	Bob	Williams	ECE	2026

major	enrollment
ECE	300

grad_yr	enrollment
2026	1500
2027	1600

Today

- Background on databases
- More on isolation
  - MVCC and snapshot isolation
- More on sharding
  - Data distribution strategies