Sending Hints

\[ R_k \bowtie_B S_i \]

**Strategy 3: Bloom Filters**

**Node 1**

<1,A>
<2,B>
<2,C>
<3,D>
<4,E>

**Node 2**

<2,X>
<3,Y>
<6,Y>
Sending Hints

\[ R_k \bowtie_B S_i \]

**Strategy 3: Bloom Filters**

Node 1
- <1,A>
- <2,B>
- <2,C>
- <3,D>
- <4,E>

Node 2
- <2,X>
- <3,Y>
- <6,Y>

Send me rows with a ‘B’ in the bloom filter summarizing the set \{2,3,6\}
Sending Hints

\[ R_k \bowtie_B S_i \]

**Strategy 3: Bloom Filters**

This is called a **bloom-join**.

Node 1
- \(<1,A>\>
- \(<2,B>\>
- \(<2,C>\>
- \(<3,D>\>
- \(<4,E>\>

Node 2
- \(<2,X>\>
- \(<3,Y>\>
- \(<6,Y>\>

Send me rows with a ‘B’ in the bloom filter summarizing the set \(\{2,3,6\}\).
Bloom Filter Construction

Empty Filter (Size: \( m = 20 \))

00000000000000000000

Use hash functions to pick a fixed number of bits (\( k = 3 \))

\[ h_1(X) = 13; \quad h_2(X) = 2; \quad h_3(X) = 5 \]

Set those bits to 1

00100100000001000000
Bloom Filter Lookup

Key 1  00101010
Key 2  01000110
Key 3  10000110
Key 4  01001100

Filters are combined by Bitwise-OR

e.g. (Key 1 | Key 2) = 01101110

How do we test for inclusion?

(Key & Filter) == Key?

(Key 1 & S) = 00101010  ✓
(Key 3 & S) = 00000110  ✗
(Key 4 & S) = 01001100  ✓

False Positive
Bloom Filter Parameters

\[ m = \text{size of the bit vector} \]
Bigger – More space used
Smaller – More false positives

\[ k = \text{# of bits set per element} \]
More Bits – More false positives
Fewer Bits – More false positives
(Need to balance #)
How do we pick M and K?
Bloom Filters

Probability that 1 bit is set by 1 hash fn

\[ \frac{1}{m} \]
Bloom Filters

Probability that 1 bit is not set by 1 hash fn

\[ 1 - \frac{1}{m} \]
Bloom Filters

Probability that 1 bit is not set by k hash fns

\[(1 - \frac{1}{m})^k\]
Bloom Filters

Probability that 1 bit is not set by k hash fns for n records

\[
(1 - \frac{1}{m})^k n
\]

So for an arbitrary record, what is the probability that all of its bits will be set?
Bloom Filters

Probability that 1 bit is set by $k$ hash fns for $n$ records

$$1 - (1 - \frac{1}{m})^{kn}$$
Bloom Filters

Probability that all k bits are set by k hash fns for n records

\[ \approx \left( 1 - \left( 1 - \frac{1}{m} \right)^{kn} \right)^k \]

\[ \approx \left( 1 - e^{-kn/m} \right)^k \]
Bloom Filters

Minimal P[collision] is at $k \approx 0.7 \cdot \frac{m}{n}$

<table>
<thead>
<tr>
<th>$m/n$</th>
<th>$k$</th>
<th>p(collision)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>~9.2%</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>~0.85%</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>~0.007%</td>
</tr>
<tr>
<td>30</td>
<td>21</td>
<td>~0.000055%</td>
</tr>
</tbody>
</table>
Bloom Filters

Minimal $P[\text{collision}]$

- $m/n = 5$: $y = (1 - e^{-1/5})^x$, $p(\text{collision}) \approx 9.2\%$
- $m/n = 10$: $y = (1 - e^{-1/10})^x$, $p(\text{collision}) \approx 0.85\%$
- $m/n = 20$: $y = (1 - e^{-1/20})^x$, $p(\text{collision}) \approx 0.007\%$
- $m/n = 30$: $y = (1 - e^{-1/30})^x$, $p(\text{collision}) \approx 0.000055\%$

5 bits/record, 3 bits set = 10% chance of collision
Parallelizing

OLAP - Parallel Queries

OLTP - Parallel Updates
Parallelism Models

Option 4: “Shared Nothing” in which all communication is explicit.

We’ll be talking about “shared nothing” for updates. Other models are easier to work with.
Data Parallelism

Replication

Partitioning

(needed for safety)
Updates

What can go wrong?

• Non-Serializable Schedules

Node 1

T1: W(X)
T2: Y(X)
T2: W(Y)
T1: W(Y)
Updates (in Parallel)

What can go wrong?

• Non-Serializable Schedules
Updates (in Parallel)

What can go wrong?

• Non-Serializable Schedules

• One Compute Node Fails
Updates (in Parallel)

What can go wrong?

• Non-Serializable Schedules
• One Compute Node Fails
• A Communication Channel Fails
• Messages delivered out-of-order
Updates (in Parallel)

What can go wrong?

• Non-Serializable Schedules
• One Compute Node Fails
• A Communication Channel Fails
• Messages delivered out-of-order

Classical Xacts
“Partitions”
Consensus
Data Parallelism

Replication

Partitioning

(needed for safety)
Simple Consensus

“Safe” … but Node 1 is a bottleneck.
Simpl-ish Consensus

Node 2 agrees to Node 1’s order for A.
Node 1 agrees to Node 2’s order for B.
Partitions

Channel Failure

Node 1

Node 2

Node Failure

Node 1

Node 2

From Node 1’s perspective, how are these cases different?
They’re not!
Failure Recovery

• Node Failure
  • The node restarts and resumes serving requests.

• Channel Failure
  • Node 1 and Node 2 regain connectivity.
Partitions

Node 1

A=1
B=5

Node 2

A=1
B=5
Partitions

Option 1: Node 1 takes over

A=1
B=5
Partitions

A=1
B=5

Node 1 takes over

Option 1: Node 1 takes over

Node 2 is down.
I control A & B now!
Partitions

Option 1: Node 1 takes over

Node 2 is down. I control A & B now!

A = 2  
B = 6
Partitions

A=2
B=6

Option 1: Node 1 takes over
Partitions

A=1
B=5

Option 1: Node 1 takes over

A=1
B=5
Partitions

Option 1: Node 1 takes over

A=1
B=5

Node 2 is down.
I control A & B now!

A=1
B=5

Node 1

Node 2
Partitions

**Option 1:** Node 1 takes over

Node 2 is down.
I control A & B now!

A = 2
B = 6
Partitions

**Option 1:** Node 1 takes over

Node 1

A=2
B=6

Node 2

A=1
B=5

INCONSISTENCY!
Partitions

Option 2: Wait

Node 1

Node 2
Partitions

Option 2: Wait

Node 1

A = 2
B = 6

Node 2
Partitions

Option 2: Wait

Node 1

I can’t talk to Node 2
Let me wait!

A = 2
B = 6

Node 2
Partitions

Option 2: Wait

Node 1

I can’t talk to Node 2
Let me wait!

Node 2

A = 2
B = 6
Partitions

Option 2: Wait

Node 1

I can’t talk to Node 2
Let me wait!

All set

Node 2

A = 2
B = 6
Partitions

Option 2: Wait
Partitions

Option 2: Wait

Node 1

I can’t talk to Node 2
Let me wait!

A = 2
B = 6
Partitions

Option 2: Wait

Node 1

I can’t talk to Node 2
Let me wait!

Still waiting…

A = 2
B = 6
Partitions

**Option 1**: Assume Node Failure

All data is available… but at risk of inconsistency.

**Option 2**: Assume Connection Failure

All data is consistent… but unavailable
Traditionally: Pick any 2
Simpl-ish Consensus

Node 1 agrees to Node 2’s order for B.
Node 2 agrees to Node 1’s order for A.
Simpl-ish Consensus

What if we need to coordinate between A & B?
Naive Commit

Coordinator

Node 1

Node 2

W(A,B)

Safe to Commit?

ACK

ACK

Safe to Commit?
That packet sure does look tasty…
Naive Commit

Coordinator  Node 1  Node 2

W(A,B) -> ACK

Is it safe to abort?
Naive Commit

Coordinator \( \rightarrow \) Node 1 \( \rightarrow \) Node 2

\[ W(A,B) \]

ACK \( \rightarrow \) ACK

What now?
How do we know Node 2 even still exists?
2-Phase Commit

- One site selected as a coordinator.
  - Initiates the 2-phase commit process.
- Remaining sites are subordinates.

- Only one coordinator per xact.
  - Different xacts may have different coordinators.
2-Phase Commit

- Coordinator sends ‘prepare’ to each subordinate.
- When subordinate receives ‘prepare’, it makes a final decision: Commit or Abort.
- The transaction is treated as if it committed for conflict detection.
- The subordinate logs ‘prepare’, or ‘abort’
- The subordinate responds ‘yes’, or ‘no’
2-Phase Commit

• If coordinator receives ‘no’ from any subordinate, it tells subordinates to ‘abort’.

• Can treat timeouts as ‘no’s

• If coordinator receives ‘yes’ from all subordinates, it tells subordinates to ‘commit’

• In both cases, the coordinator first logs the decision and forces the log to local storage.
2-Phase Commit

• Subordinates perform abort or commit as appropriate (logging as in single-site ARIES)
• Subordinates ‘ack’nowledge the coordinator.
• The transaction is complete once the coordinator receives all ‘acks’.
2PC for Replication

- Optimization: We don’t need 100% responses from replicas.
- Replicas can be reconstructed from others.
- Asserting ‘preparedness’ can be difficult.
- How much failure tolerance do we want?
- We can tolerate N failures by waiting for N+1 responses during the ‘prepare’ phase.
Recovery

How do we recover from a (transient) coordinator crash in Phase 1?

What information/communication state is lost?

Can it be recovered?

(Does it need to be?)
Recovery

How do we recover from a (transient) coordinator crash in Phase 2?

What information/communication state is lost?

Can it be recovered?
Recovery

How do we recover from a (transient) subordinate crash in Phase 1?

What information/communication state is lost?
Can it be recovered?
Recovery

How do we recover from a (transient) subordinate crash in Phase 2?

What information/communication state is lost?
Can it be recovered?