Update Parallelism
Parallelism Models

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

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

Replication

Partitioning

(needed for safety)
Updates

What can go wrong?

- Non-Serializable Schedules
Updates (in Parallel)

What can go wrong?

• Non-Serializable Schedules
Updates (in Parallel)

What can go wrong?

- Non-Serializable Schedules
- One Compute Node Fails

Node 1

Node 2
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
  
  (needed for safety)

- Partitioning
Simple Consensus

Primary

Node 1

X Y

A B

Secondary

Node 2

X Y

A B

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

Node 1

Primary for A

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

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

Primary for B

Node 2

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 Failure

Node 1

From Node 1’s perspective, these are the same!
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

Node 1

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

Partitions

Option 1: Node 1 takes over

A=1
B=5
Partitions

Option 1: Node 1 takes over

A=2
B=6

Node 1

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

A = 2
B = 6

Node 2
Partitions

Option 1: Node 1 takes over

Node 1

A=2
B=6

Node 2
Partitions

Option 1: Node 1 takes over

Node 1

A=1
B=5

Node 2

A=1
B=5
Partitions

Option 1: Node 1 takes over

Node 1

Node 2

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

Option 1: Node 1 takes over

INCONSISTENCY!
Partitions

Option 2: Wait

Node 1

Node 2
Partitions

Option 2: Wait

Node 1

Node 2

A = 2
B = 6
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

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!

All set

Node 2

A = 2
B = 6
Partitions

Option 2: Wait

Node 1

Node 2
Partitions

Option 2: Wait

Node 1

I can’t talk to Node 2
Let me wait!

A = 2
B = 6
Partitions

Option 2: Wait

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
I prefer this phrasing.
Simpl-ish Consensus

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

W(A,B) → ACK  → ACK

What now?
Naive Commit

Coordinator  Node 1  Node 2

W(A)

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.
Assumptions

• Undo/Redo Logging at Participants
  • Participants can Abort an Xact at any time
  • Participants can recover from a crash
• Redo Logging at Coordinator
  • Coordinator can recover from a crash
• All logs replicated (to recover from hard failures)
## Phase 1 - Prepare

<table>
<thead>
<tr>
<th>Coordinator</th>
<th>Node 1</th>
<th>Node 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Phase 1 - Prepare

Coordinator | Node 1 | Node 2
---|---|---
“Prepare”
Phase 1 - Prepare

Coordinator  Node 1  Node 2

“Prepare”  “Commit”  “Commit”
Phase 1 - Prepare

Coordinator

"Prepare"

Node 1

"Commit"

Node 2

"Commit"

We are go for Commit
Phase 2 - Commit

Coordinator

"Prepare"

We are go for Commit

Node 1

"Commit"

Node 2

"Commit"

"Commit"
Phase 2 - Commit

"Prepare"

We are go for Commit

"Commit"

ACKs received
Commit successful

"Commit"

"Commit"

"ACK"

"ACK"
If any participant aborts in Phase 1, everyone aborts.
Guarantees

A Node “Commit” means the node is able to commit. A Coordinator “Commit” means the transaction must commit.
Guarantees

Once a node commits, the xact is still not committed yet. However the node must avoid breaking the commit.
Failure Modes

Coordinator

Node 1

Node 2

“Prepare”

“Commit”

Prepare unreceived and unacknowledged: Coordinator (1) Retries, or (2) Aborts
Failure Modes

Coordinator

“Prepare”

Node 1

“Commit”

Node 2

CRASH!

Node 2 crashes before responding: Restart and continue as a dropped packet
Node “Commit” unreceived: (1) Re-sent “Prepare” can be ignored. (2) Node still able to abort.
Node 2 crashes after responding: Restart from log
Failure Cases

Coordinator

„Prepare“

Node 1

„Commit“

Node 2

„Commit“

We are go for Commit

„Commit“

„ACK“

Coordinator “Commit” unreceived: Commit must happen, coordinator resends
Failure Cases

Coordinator

“Prepare”

We are go for Commit

Node 1

“Commit”

Node 2

“Commit”

“Commit”

“ACK”

CRASH!

Node 2 crash: Restart. Already logged “Commit” message, so all is well.
Failure Cases

Coordinator  Node 1  Node 2

"Prepare"

We are go for Commit

"Commit"

"Commit"

"Commit"

"ACK"

"ACK"

Node "Ack" unreceived: Ok. Resent "Commit" ignored by node
Node crash after “Ack”: Ok. Log already recorded commit
Replication

- **Mode 1**: Periodic Backups
  - Copy the replicated data nightly/hourly.

- **Mode 2**: Log Shipping
  - Only send changes (replica serves as the log).
Replication

- Ensuring durability
- Ensuring write-consistency under 2PC
- Ensuring read-consistency without 2PC
Ensuring Durability

When is a replica write durable?
Ensuring Durability

Never.

What you should be asking is how much durability do you need?
Ensuring Durability

For $\mathbf{N}$ Failures
$\mathbf{N+1}$ Replicas

(Assuming Failure $=$ Crash)
Ensuring Write Consistency

Coordinator

"Prepare"

Node 1

"Commit"

Node 1 asserts that the commit is durable!

What if Node 1 fails?
Ensuring Write Consistency

Waiting for Node 1 to replicate is sloooooow!
Let the coordinator take over!
Ensuring Write Consistency

Like 2PC...
... but better. We may not need to wait for the replica
Ensuring Write-Consistency

Coordinator Alice

A: Prepare

Replica 1

Replica 2

Replica 3

Coordinator Bob

B: Prepare
Ensuring Write-Consistency

Coordinator Alice

B: Prepare
A: Prepare

Replica 1

B: Prepare
A: Prepare

Replica 2

B: Prepare
A: Prepare

Replica 3

Coordinator Bob
Ensuring Write-Consistency

Coordinator Alice

B: Prepare
Commit!

Replica 1

Coordinator Bob

B: Prepare
Commit!

Replica 2

A: Prepare
Commit!

Replica 3
Ensuring Write-Consistency

Majority Vote

$N$ Replicas

$\left(\frac{N}{2}\right)+1$ Votes Needed
Ensuring Read Consistency

Forget transactions, let’s go back to reads & writes.

Can we do better than 2PC if we don’t need xacts?
(1) Alice writes ‘A’
(1) Alice writes ‘A’

W(A = 3)

(2) Alice tells Bob
Replica 1

(1) Alice writes ‘A’
W(A = 3)

(2) Alice tells Bob

Replica 2

Replica 3

(3) Bob reads ‘A’
R(A)
(1) Alice writes ‘A’

(2) Alice tells Bob

W(A = 42) → Replica 1

R(A) → (3) Bob reads ‘A’

What can we do to guarantee that Bob will see the 42?
Ensuring Read Consistency

**Approach:** Alice and Bob each wait for multiple responses.

- Alice waits for ‘ack’s
- Bob waits for read responses.

How many responses are required for each?
Replica 1

ACK

W(A = 42)

Replica 2

Replica 3

R(A)
Replica 1

W(A = 42)

ACK

Replica 2

R(A)

Replica 3

“666”
Replica 1

ACK

W(A = 42)

Replica 2

R(A)

“666”

Replica 3
$W(A = 42)$

ACK

Replica 1

$R(A)$

“666”

Replica 3

Replica 2
Ensuring Read-Consistency

Like Majority Vote

N Replicas
R Replica Reads Needed
W Writer Acks Needed
R + W > N