Update Parallelism

April 30, 2018
HW 3 Posted
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

Node 1

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

What can go wrong?

• Non-Serializable Schedules
• One Compute Node Fails
• A Communication Channel 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
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
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”
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

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
Partitions

Option 1: Node 1 takes over

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

A=1
B=5
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

A=2
B=6
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 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
Partitions

Option 2: Wait

Node 1

A = 2
B = 6

Node 2
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!

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
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
<table>
<thead>
<tr>
<th>C</th>
<th>A</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>V</td>
<td>A</td>
</tr>
<tr>
<td>N</td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>S</td>
<td>A</td>
<td>T</td>
</tr>
<tr>
<td>I</td>
<td>T</td>
<td>I</td>
</tr>
<tr>
<td>S</td>
<td>I</td>
<td>T</td>
</tr>
<tr>
<td>I</td>
<td>O</td>
<td>N</td>
</tr>
<tr>
<td>C</td>
<td>L</td>
<td>T</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>S</td>
</tr>
</tbody>
</table>

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

What if we need to coordinate between A & B?

Master for A

Node 1

Master for B

Node 2

Withdraw $1000 from A
Deposit $1000 into B

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

Coordinator       Node 1       Node 2

W(A,B)
Naive Commit

Coordinator  Node 1  Node 2

\(W(A,B)\)

Safe to Commit?
Naive Commit

Coordinator \rightarrow Node 1 \rightarrow Node 2

W(A,B) \rightarrow \text{ACK} \rightarrow \text{Safe to Commit?}
Naive Commit

Coordinator \rightarrow Node 1 \rightarrow Node 2

W(A,B) \rightarrow ACK

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

Coordinator \[W(A, B)\] \rightarrow Node 1 \rightarrow Node 2 \rightarrow ACK

Is it safe to abort?
Naive Commit

Coordinator  Node 1  Node 2

W(A,B)  ACK  ACK

What now?
Naive Commit

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

"Prepare"

Node 1

"Commit"

Node 2

"Commit"
Phase 1 - Prepare

Coordinator

Node 1

Node 2

“We are go for Commit”

“Prepare”

“Commit”

“Commit”
Phase 2 - Commit

Coordinator

Node 1

Node 2

“Prepare”

“Commit”

“We are go for Commit”

“Commit”

“Commit”
Phase 2 - Commit

Coordinator

"Prepare"

Node 1

We are go for Commit

Node 2

"Commit"

"Commit"

"Commit"

"ACK"

"ACK"

ACKs received

Commit successful

Commit successful
Aborting

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”
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
Failure Modes

Coordinator

"Prepare"

Node 1

"Commit"

Node 2

"Commit"
Node “Commit” unreceived: (1) Re-sent “Prepare” can be ignored. 
(2) Node still able to abort.
Failure Modes

Coordinator

Node 1

Node 2

“Prepare”

“Commit”

“Commit”

CRASH!
Node 2 crashes after responding: Restart from log
Failure Cases

Coordinator

"Prepare"

Node 1

We are go for Commit

"Commit"

Node 2

"Commit"

"Commit"

"ACK"
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"

Node 1

"Commit"

Node 2

"Commit"

"Commit"

"ACK"

We are go for Commit

CRASH!
Failure Cases

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

<table>
<thead>
<tr>
<th>Coordinator</th>
<th>Node 1</th>
<th>Node 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Prepare”</td>
<td>“Commit”</td>
<td>“Commit”</td>
</tr>
<tr>
<td>We are go for Commit</td>
<td>“ACK”</td>
<td>“ACK”</td>
</tr>
</tbody>
</table>

We are go for Commit
Failure Cases

Coordinator  Node 1  Node 2

"Prepare"  "Commit"  "Commit"

We are go for Commit

"Commit"  "ACK"  "ACK"

Node "Ack" unreceived: Ok. Resent "Commit" ignored by node
Failure Cases

Coordinator  Node 1  Node 2

“Prepare”    “Commit”    “Commit”

We are go for Commit

“Commit”    “ACK”     “ACK”

CRASH!
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

- **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.
Ensuring Durability

Never.

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

For $N$ Failures
$N+1$ 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

Coordinator

"Prepare"

Node 1

"Prepare"

Replica

"Commit"

"Commit"
Ensuring Write Consistency

Coordinator

"Prepare"

Node 1

"Prepare"

Replica

"Commit"

"Commit"

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

Coordinator

Node 1

Replica

“Prepare”

“Commit”

“Commit”
Ensuring Write Consistency

Like 2PC…

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

Coordinator Alice

A: Prepare

Coordinator Bob

B: Prepare

Replica 1

Replica 2

Replica 3
Ensuring Write-Consistency

Coordinator Alice

A: Prepare
B: Prepare

Replica 1

Coordinator Bob

A: Prepare
B: Prepare

Replica 2

A: Prepare
B: Prepare

Replica 3
Ensuring Write-Consistency

Coordinator Alice

B: Prepare

Commit!

Replica 1

Coordinator Bob

A: Prepare

Commit!

Replica 3

B: Prepare

Replica 2
Ensuring Write-Consistency

**Majority Vote**

\[ N \text{ Replicas} \]

\[ (N/2)+1 \text{ Votes Needed} \]
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’

W(A = 3)

Replica 1

Replica 2

Replica 3
(1) Alice writes ‘A’

(2) Alice tells Bob

W(A = 3)

Replica 1

Replica 2

Replica 3
(1) Alice writes ‘A’

W(A = 3)

(2) Alice tells Bob

(3) Bob reads ‘A’

R(A)
(1) Alice writes ‘A’

(2) Alice tells Bob

W(A = 42)

(3) Bob reads ‘A’

R(A)

Replica 1

Replica 2

Replica 3

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

Replica 2

W(A = 42)

Replica 3

R(A)

“666”

Replica 1

Replica 2

Replica 3

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2

Replica 3

Replica 1

Replica 2
Replica 1

Replica 2

Replica 3

W(A = 42)

R(A)

"666"

ACK
Replica 1

Replica 2

Replica 3

W(A = 42)

ACK

R(A)

"666"
Ensuring Read-Consistency

Like Majority Vote

\[ N \text{ Replicas} \]
\[ R \text{ Replica Reads Needed} \]
\[ W \text{ Writer Acks Needed} \]
\[ R + W > N \]