CLOUD COMPUTING CONCEPTS

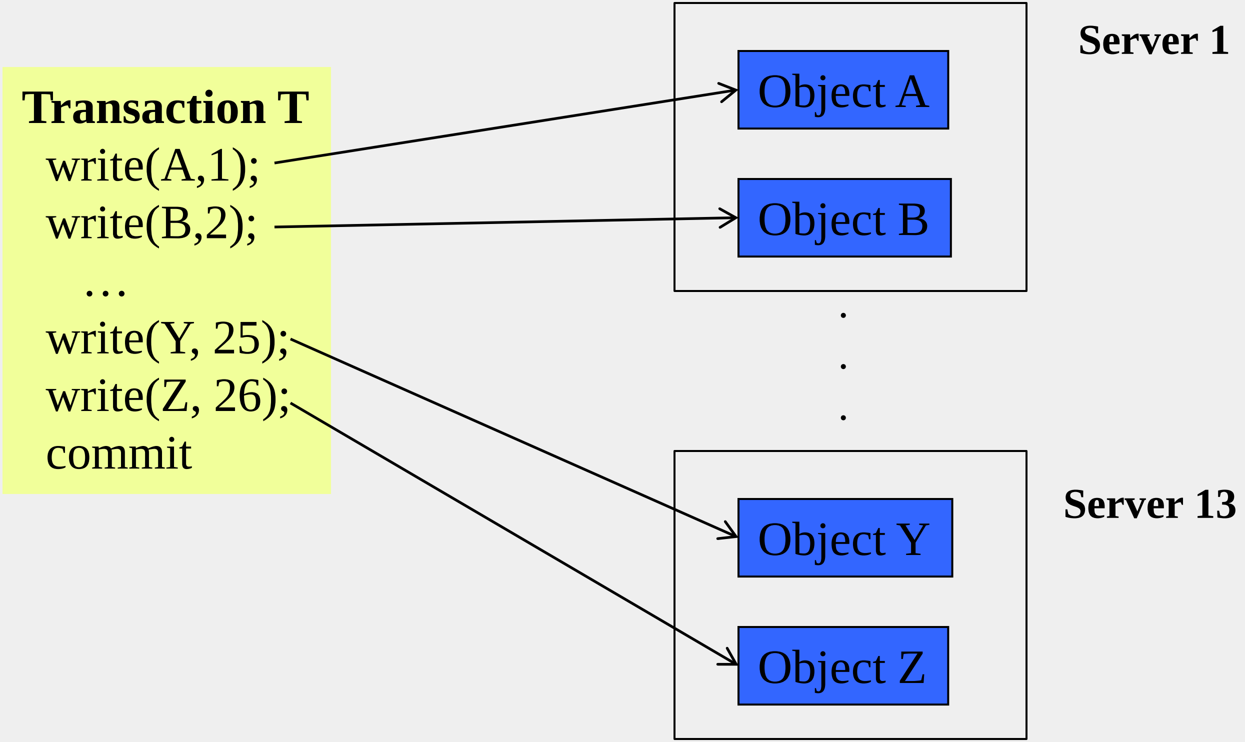
with Indranil Gupta (Indy)

REPLICATION CONTROL

Lecture B

TWO-PHASE COMMIT

Transactions with Distributed Servers



Transactions with Distributed Servers (2)

- Transaction T may touch objects that reside on different servers
- When T tries to commit
  - Need to ensure all these servers commit their updates from T => T will commit
  - Or none of these servers commit => T will abort
- What problem is this?

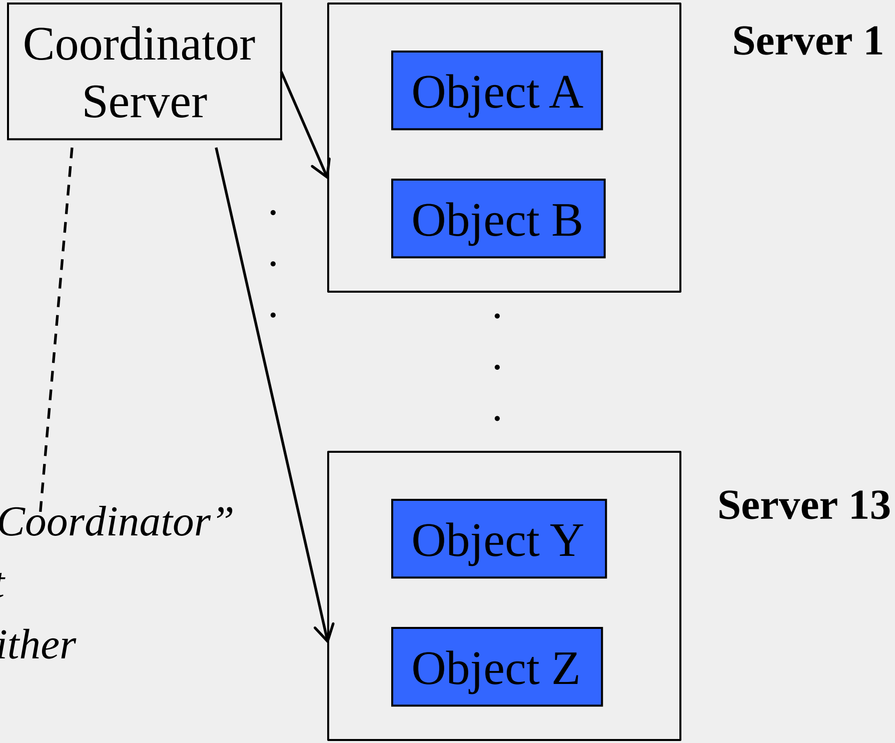
Transactions with Distributed Servers (2)

- Transaction T may touch objects that reside on different servers
- When T tries to commit
  - Need to ensure all these servers commit their updates from T => T will commit
  - Or none of these servers commit => T will abort
- What problem is this?
  - Consensus!
  - (It’s called the “Atomic Commit problem”)

One-phase Commit

Transaction T

```
write(A,1);
write(B,2);
  …
write(Y, 25);
write(Z, 26);
commit
```

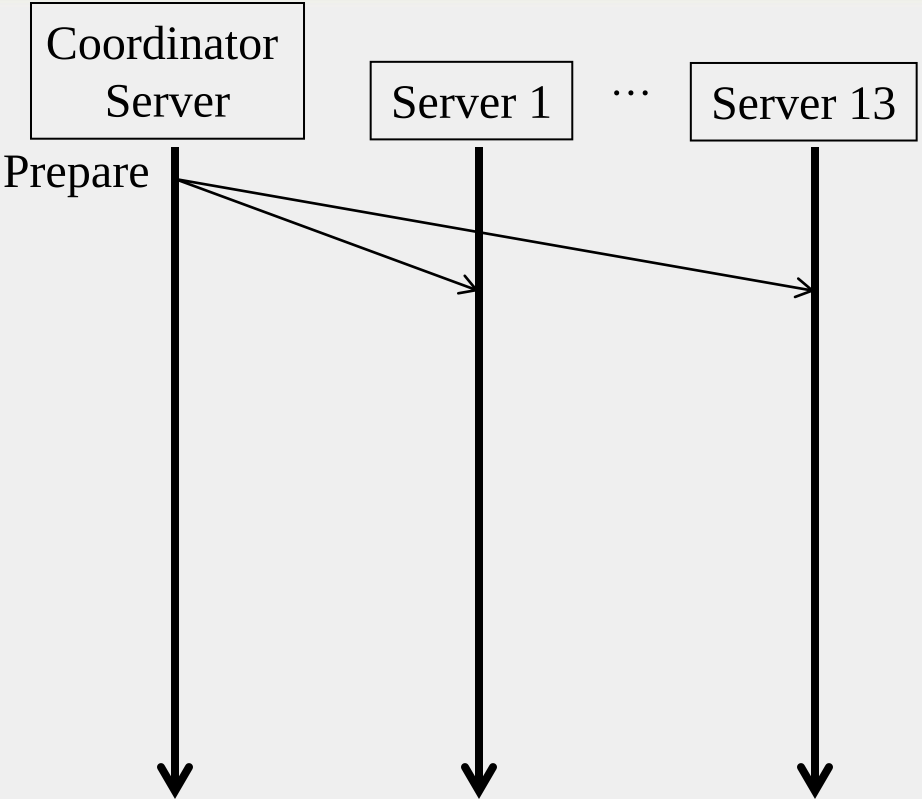


- *Special server called "Coordinator" initiates atomic commit*
- *Tells other servers to either commit or abort*

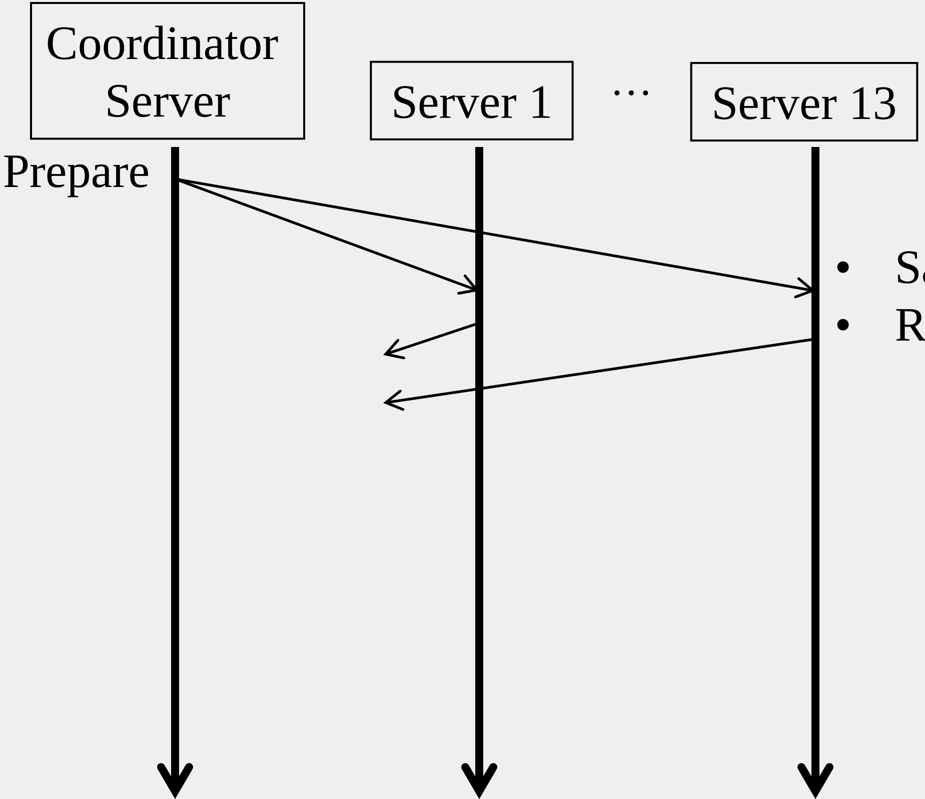
One-phase Commit: Issues

- Server with object has no say in whether transaction commits or aborts
  - If object corrupted, it cannot commit (while other servers have committed)
- Server may crash before receiving commit message, with some updates still in memory

Two-phase Commit

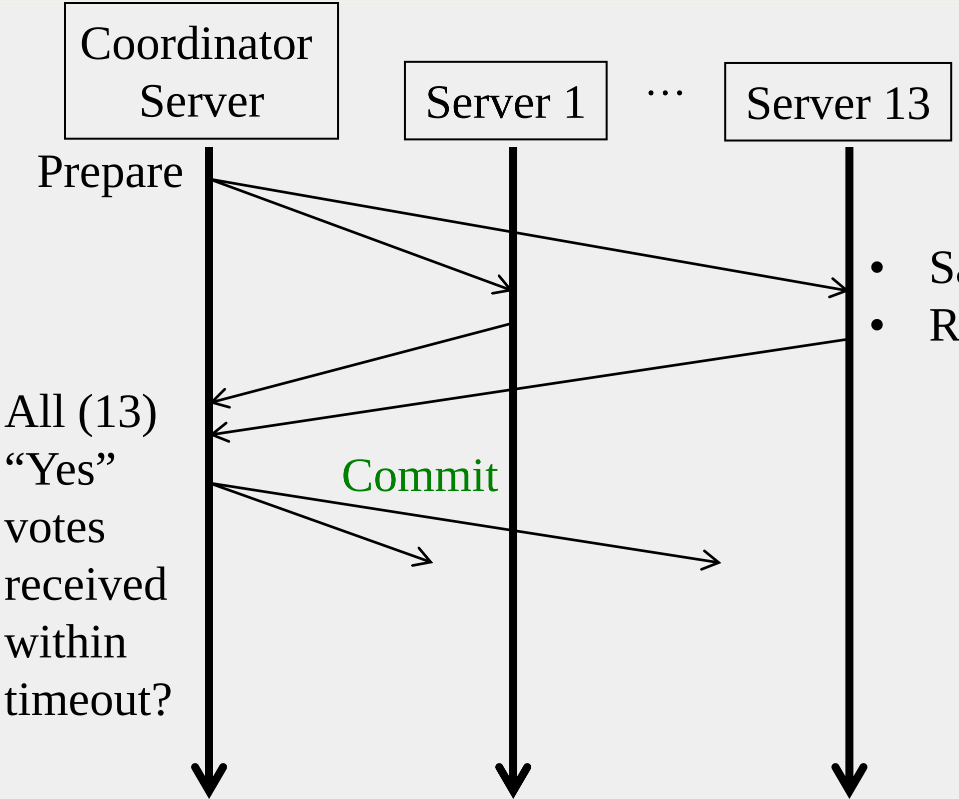


Two-phase Commit



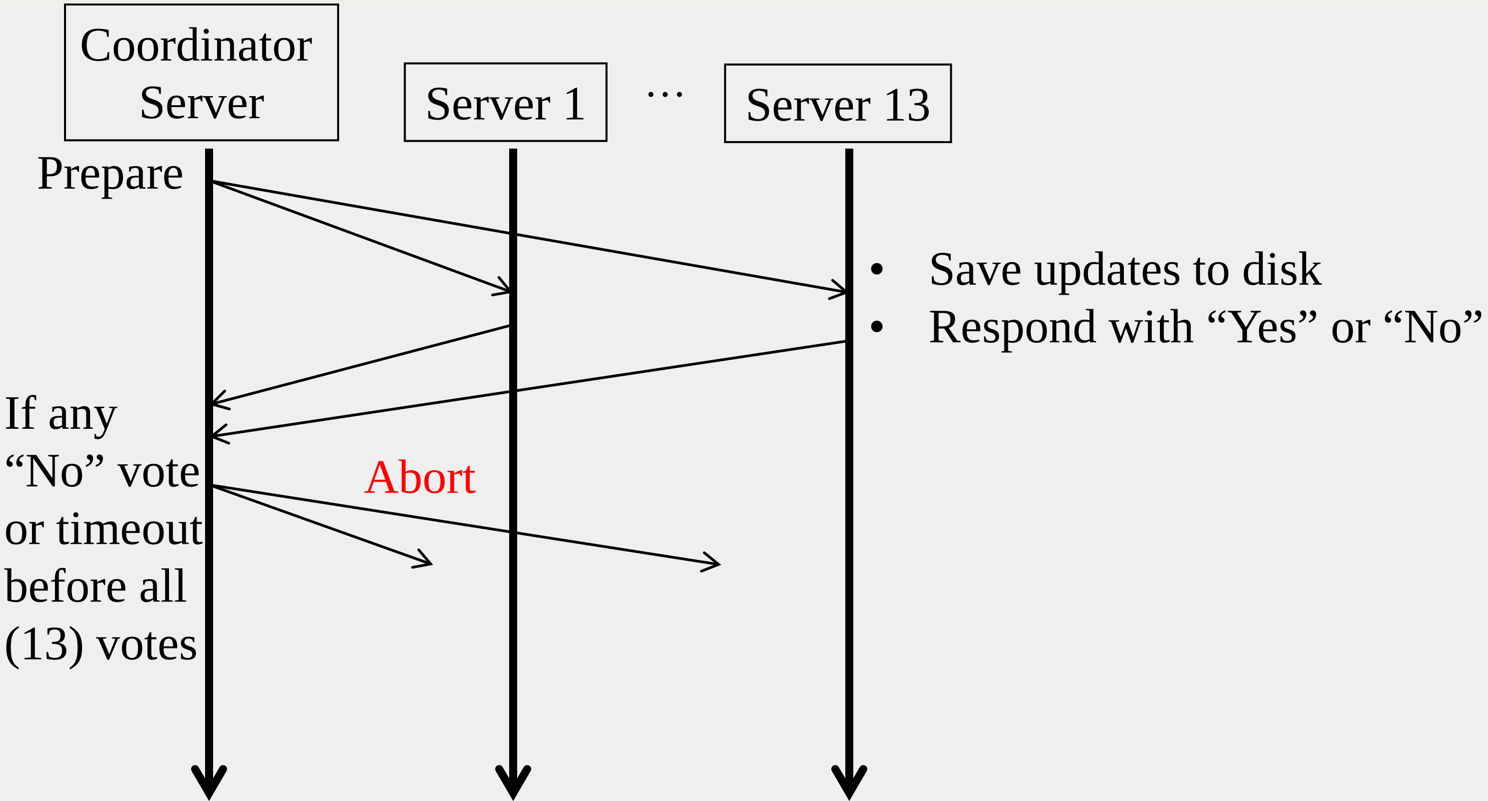
- Save updates to disk
- Respond with “Yes” or “No”

Two-phase Commit



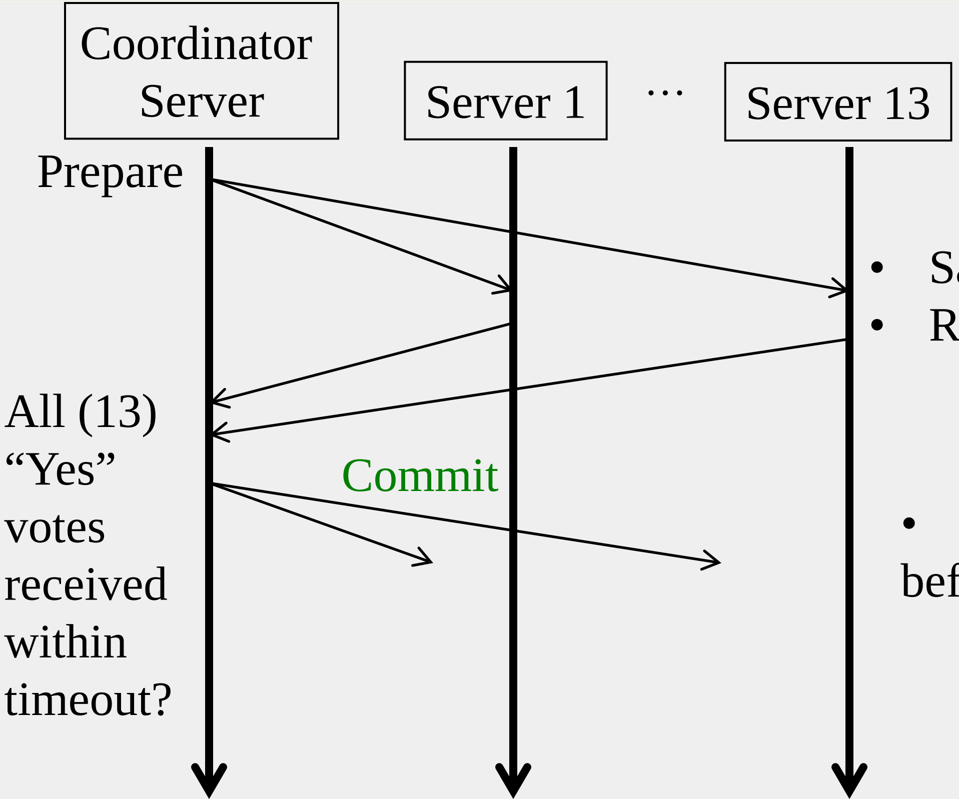
- Save updates to disk
- Respond with "Yes" or "No"

Two-phase Commit



- Save updates to disk
- Respond with “Yes” or “No”

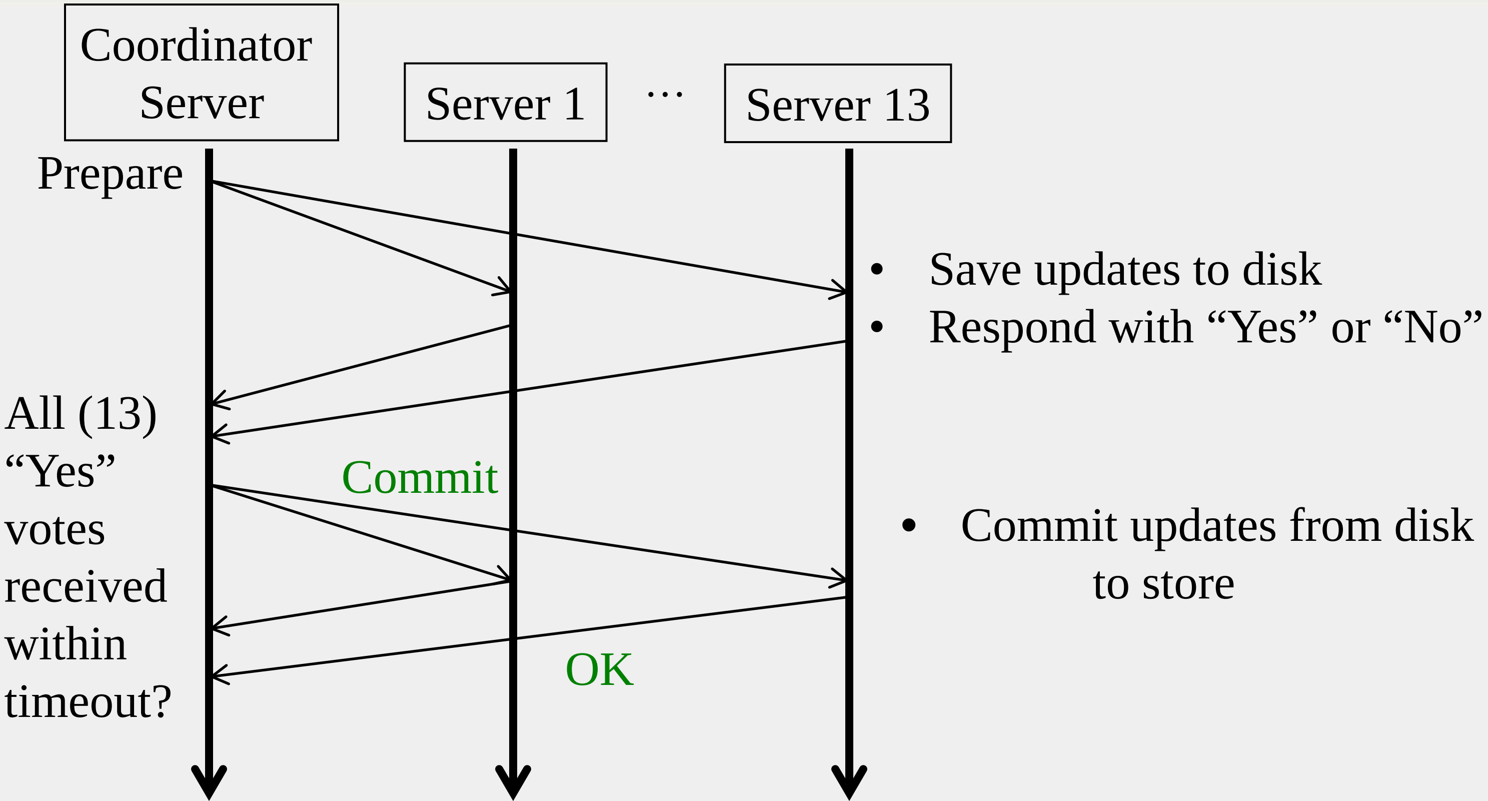
Two-phase Commit



- Save updates to disk
- Respond with “Yes” or “No”

- Wait! Can’t commit or abort before receiving next message!

Two-phase Commit



- Save updates to disk
- Respond with “Yes” or “No”

- Commit updates from disk to store

Failures in Two-phase Commit

- To deal with server crashes
  - Each server saves tentative updates into permanent storage, right before replying Yes/No in first phase. Retrievable after crash recovery.
  - Coordinator logs votes and decisions too
- To deal with coordinator crashes
  - Coordinator logs all decisions and received/sent messages on disk
  - After recovery or new election => new coordinator takes over
- To deal with Prepare message loss
  - The server may decide to abort unilaterally after a timeout for first phase (server will always vote No, and so coordinator will also eventually abort)

Failures in Two-phase Commit (2)

- To deal with Yes/No message loss, coordinator aborts the transaction after a timeout (pessimistic!). It must announce Abort message to all.
- To deal with Commit or Abort message loss
  - Server can poll coordinator (repeatedly)
- If server voted Yes, it cannot commit unilaterally before receiving Commit message
- If server voted No, can abort right away (why?)

Using Paxos in Distributed Servers

Atomic Commit

- Can also use Paxos to decide whether to commit a transaction or not
- But need to ensure that if any server votes No, everyone aborts

Ordering updates

- Paxos can also be used by replica group (for an object) to order all updates
  - Server proposes message for next sequence number
  - Group reaches consensus (or not)

SUMMARY

- Multiple servers in cloud
  - Replication for fault-tolerance
  - Load balancing across objects
- Replication flavors using concepts we learnt earlier
  - Active replication
  - Passive replication
- Transactions and distributed servers
  - Two-phase commit