Calvin: Fast Distributed Transactions for Partitioned Database Systems SIGMOD ‘12

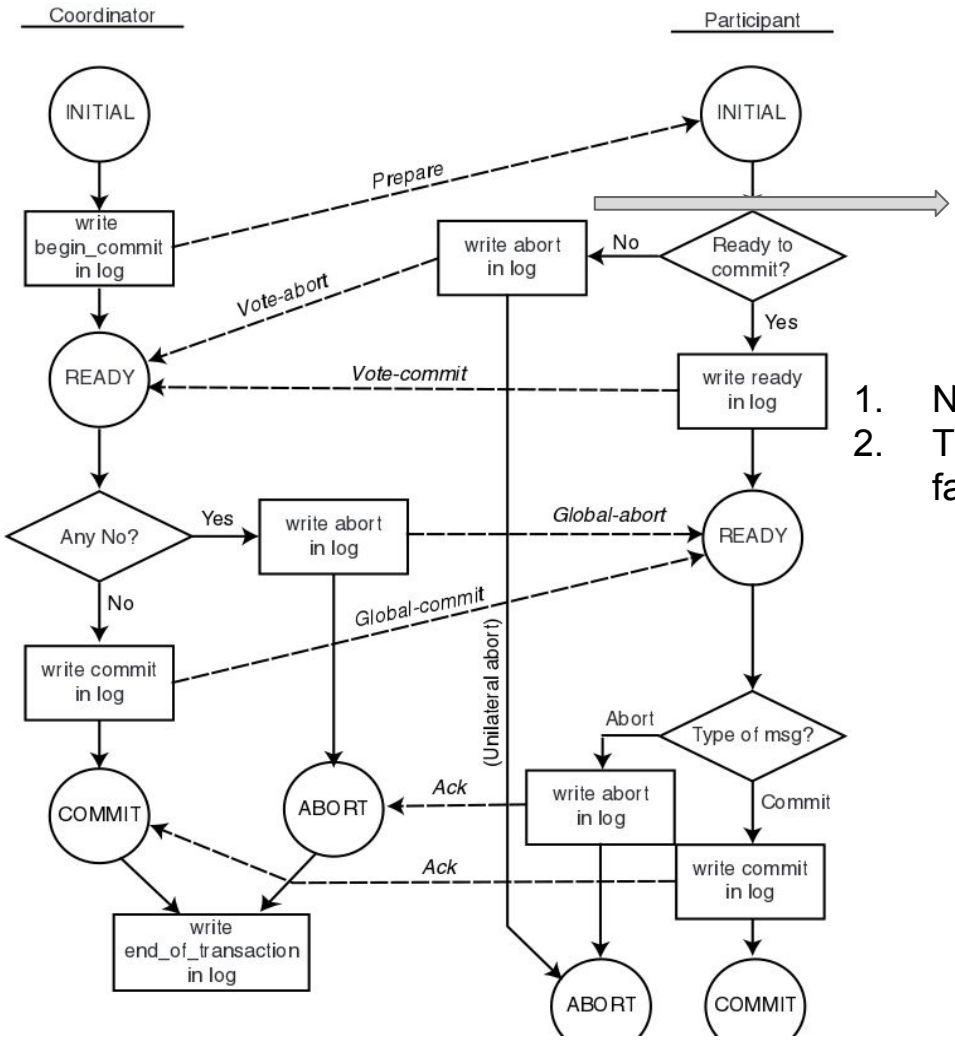
Alexander Thomson,Daniel J Abadi et al. (Yale)
Presented by Ishan Chawla

Problem statement

- Distributed transactions creates problems in Scalability
- Why?
  - Locks held in the agreement protocol of 2PC increase contention
- What is the current workaround?
  - Provide minimal transactional support to achieve scalable systems
- The authors target the primary problem in providing transactional support in scalable systems

Solution

- Maybe 2PC duration could be made shorter ?
- How ?
- The solution lies in the working of the 2PC algorithm



Why can aborts be there ?

1. Node failures
2. Transaction logic/constraint failure

Figure from Oszu et al.

Phase done away with

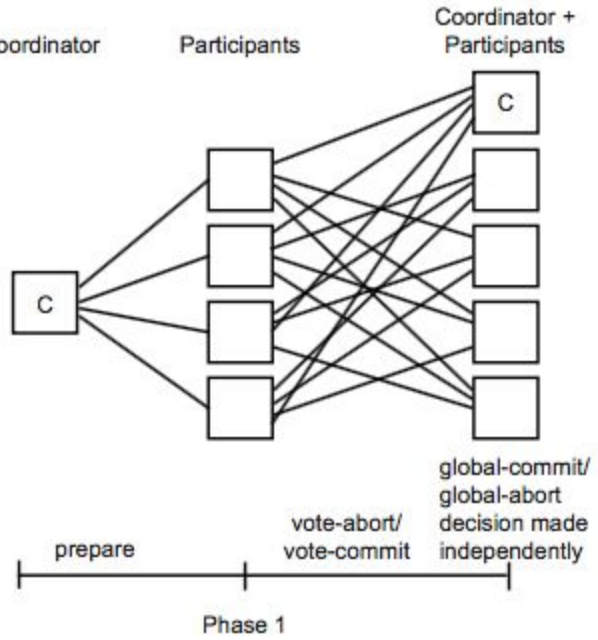


Fig. 12.13 Distributed 2PC Communication Structure

Figure from Oszu et al.

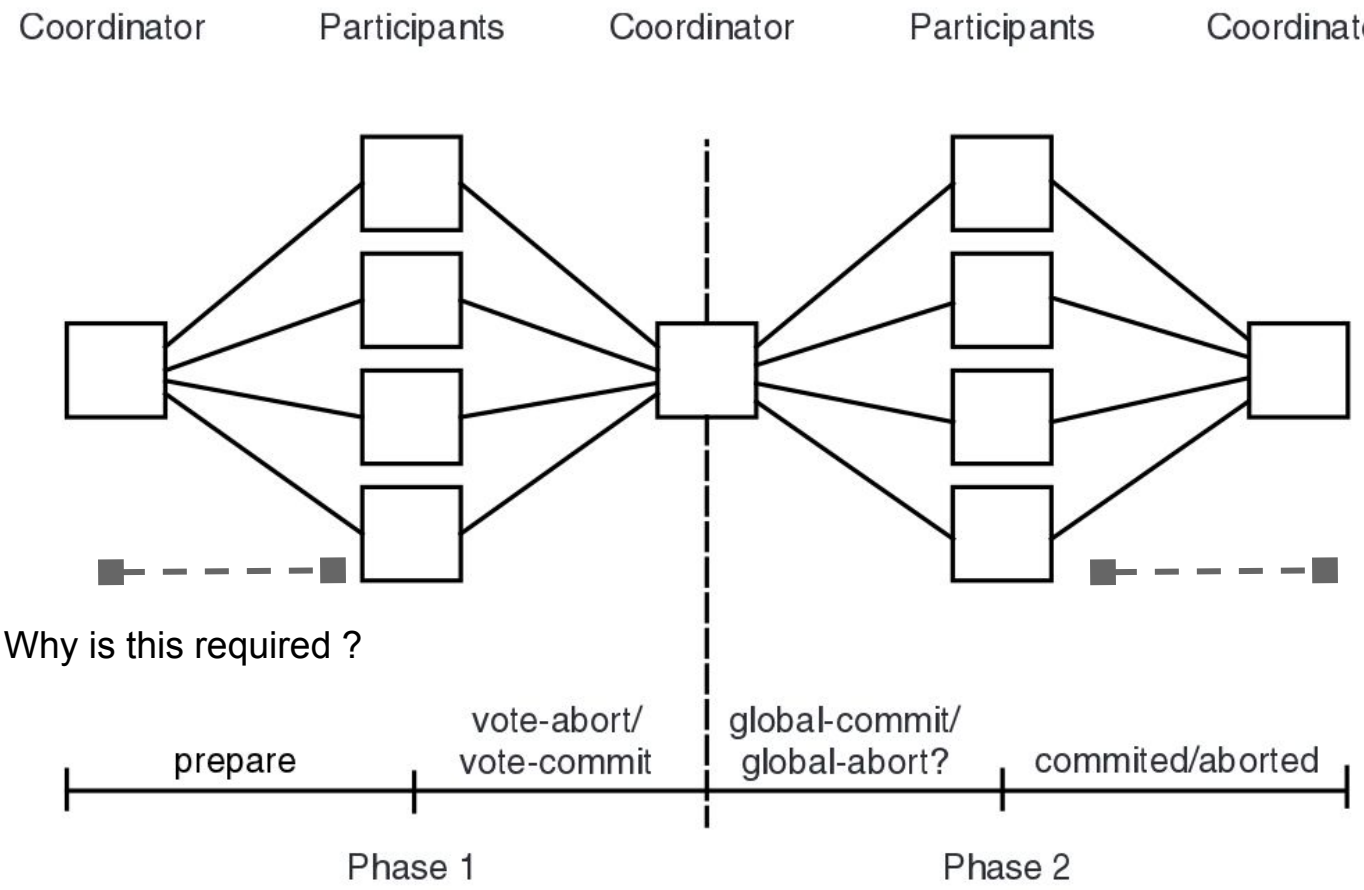


Fig. 12.11 Centralized 2PC Communication Structure

Figure from Oszu et al.

1. If no reply within some time, node has failed . The coordinator can take the abort decision after timeout

What if nodes don’t fail?

That’s unrealistic !

But what if , when some node fails it can recover to that state later no ?

But what about the unfairness to the transactions stalling system till node wakes up?

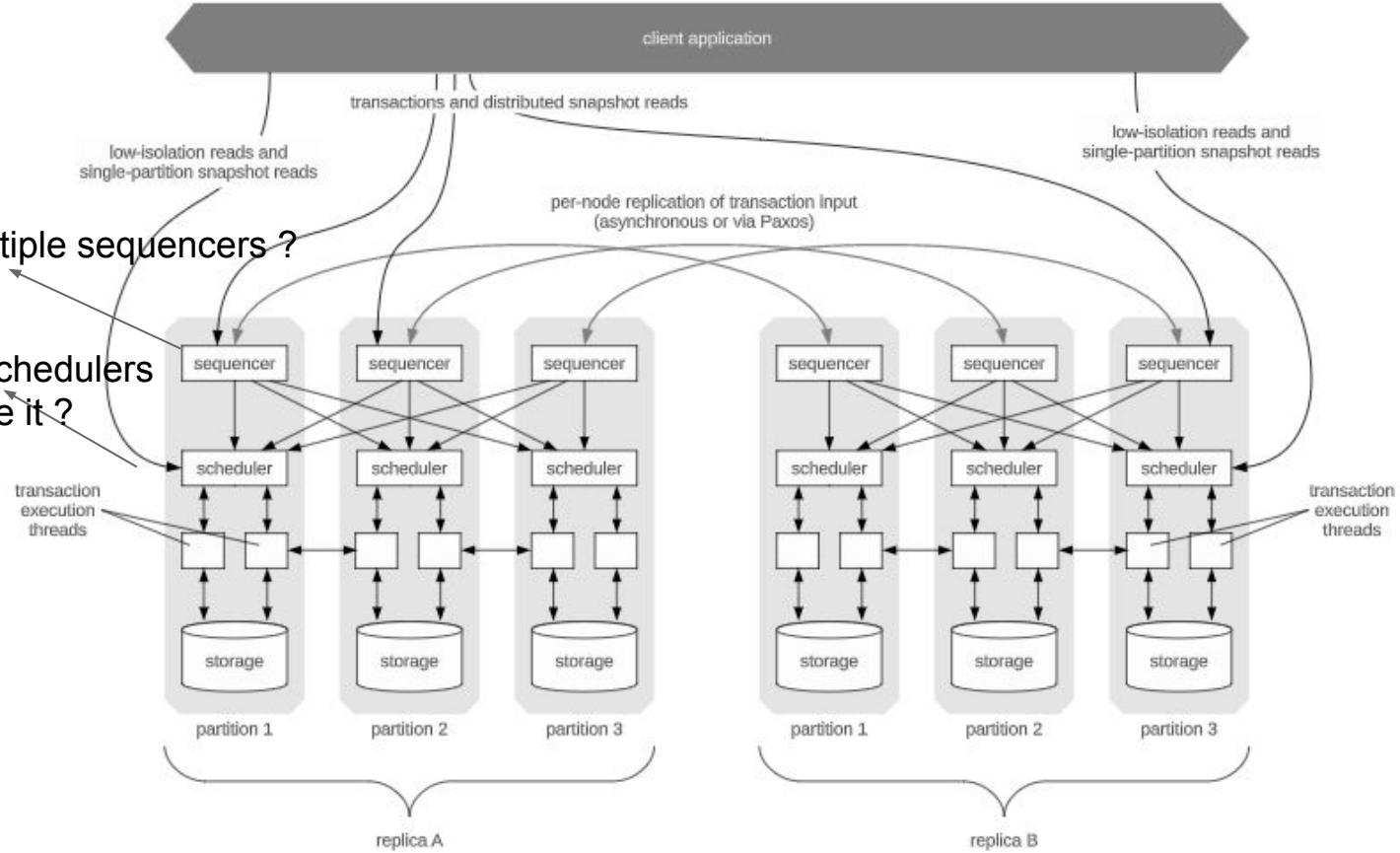
-Even if we allow the stall , another bigger problem is the non determinism

-With the same transaction inputs, an arbitrary serialisable order of transactions can

Take place, leading to a possibly different database state.

- How to avoid the stall problem ?

Add replication !



Why multiple sequencers ?

How do schedulers Not violate it ?

Figure 1: System Architecture of Calvin

How is this order ensured ?

- Deterministic lock manager
- A Thread in scheduler scans all transactions sent in epoch and acquires locks on all variables in read/write sets of transactions . Hence Transactions request all locks of its lifetime ahead of time. This is required to ensure thread scheduling cannot change the order of execution . **This is the mechanism.**

Transaction Execution - Worker Threads

1. Read/Write set Analysis - Identifies locality of Transaction variables + Active/Passive participants
2. Perform local reads
3. Serve Remote reads - Passive participants end here
4. Collect remote read results - Execute on active participants
5. Transaction logic and apply writes - Execute on active participants

Dependent Transactions

1. Mechanism of Reconnaisance query

Inexpensive, unreplicated read only query that performs all necessary reads to figure out the read/write sets

Records must be rechecked . Process needs to be deterministically restarted if read/write set not valid

Eg. Index lookups. Indices on voltatile fields like stock price are rare.

Hardware configuration

Amazon EC2 using High-CPU/Extra-Large

instances, which promise 7GB of memory and 20 EC2 Compute

Units—8 virtual cores with 2.5 EC2 Compute Units each

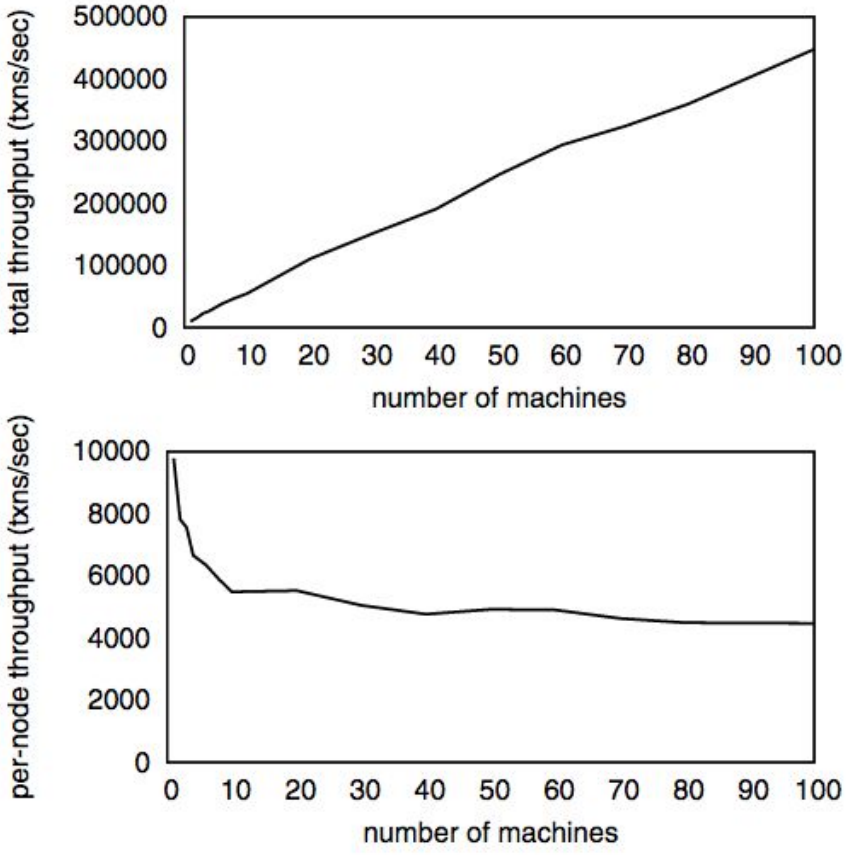


Figure 4: Total and per-node TPC-C (100% New Order) throughput, varying deployment size.

Future research scope

1. Development of seamless failover system , in which entire set of replicas need not be replaced.
2. Developing schemes in which entire read/write set need not be known .
3. Develop schemes in which determinism in conflict resolution is there to ensure a deterministic order of exectution.

Thank You !