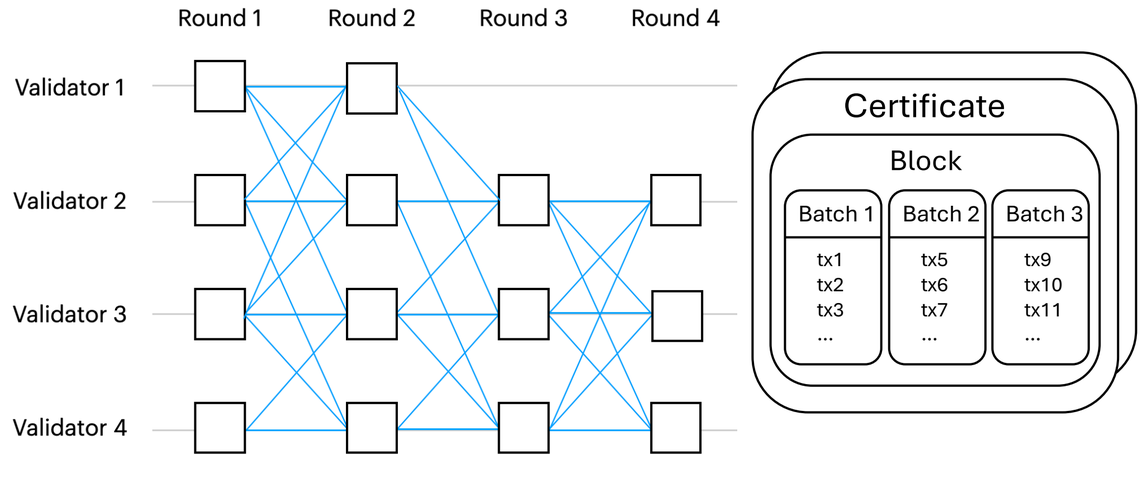
Adaptive load balancing in DAG-based consensus protocols

Zhen Ping Khor (University of Pennsylvania)* Mohammad Amiri (Stony Brook University)
Boon Thau Loo (University of Pennsylvania)

What is BFT? Byzantine Fault Tolerance (BFT) ensures that a distributed system (like a blockchain or database) remains consistent even if some nodes fail or act maliciously.
The Traditional Bottleneck: Classic protocols rely on a single leader to order transactions and broadcast data. This creates a sequential choke point that limits scalability.
The DAG Innovation: Modern DAG-based consensus protocols [1] **decouple data dissemination from ordering**. This allows nodes to propose blocks in parallel based on batch digest, drastically increasing throughput.
DAG of certificates: Unlike linear chains, each certificate is a signed block containing batches of transactions that are partially ordered, allowing for high parallel throughput before a total order is determined.

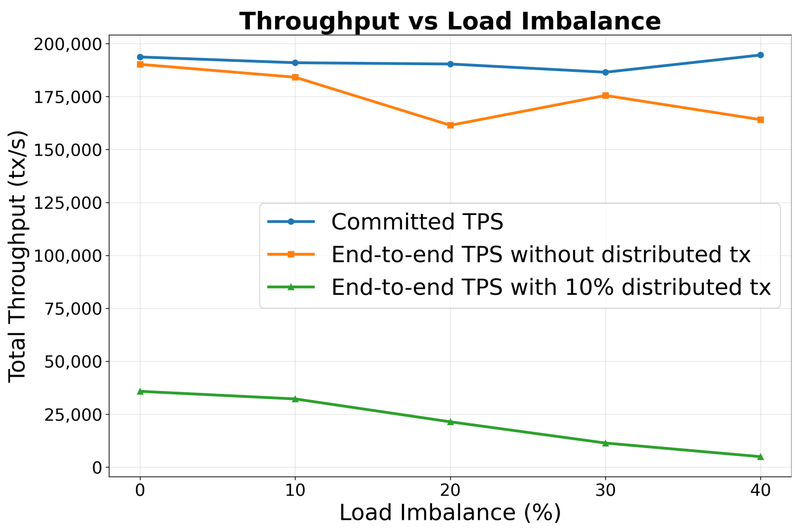


Motivation

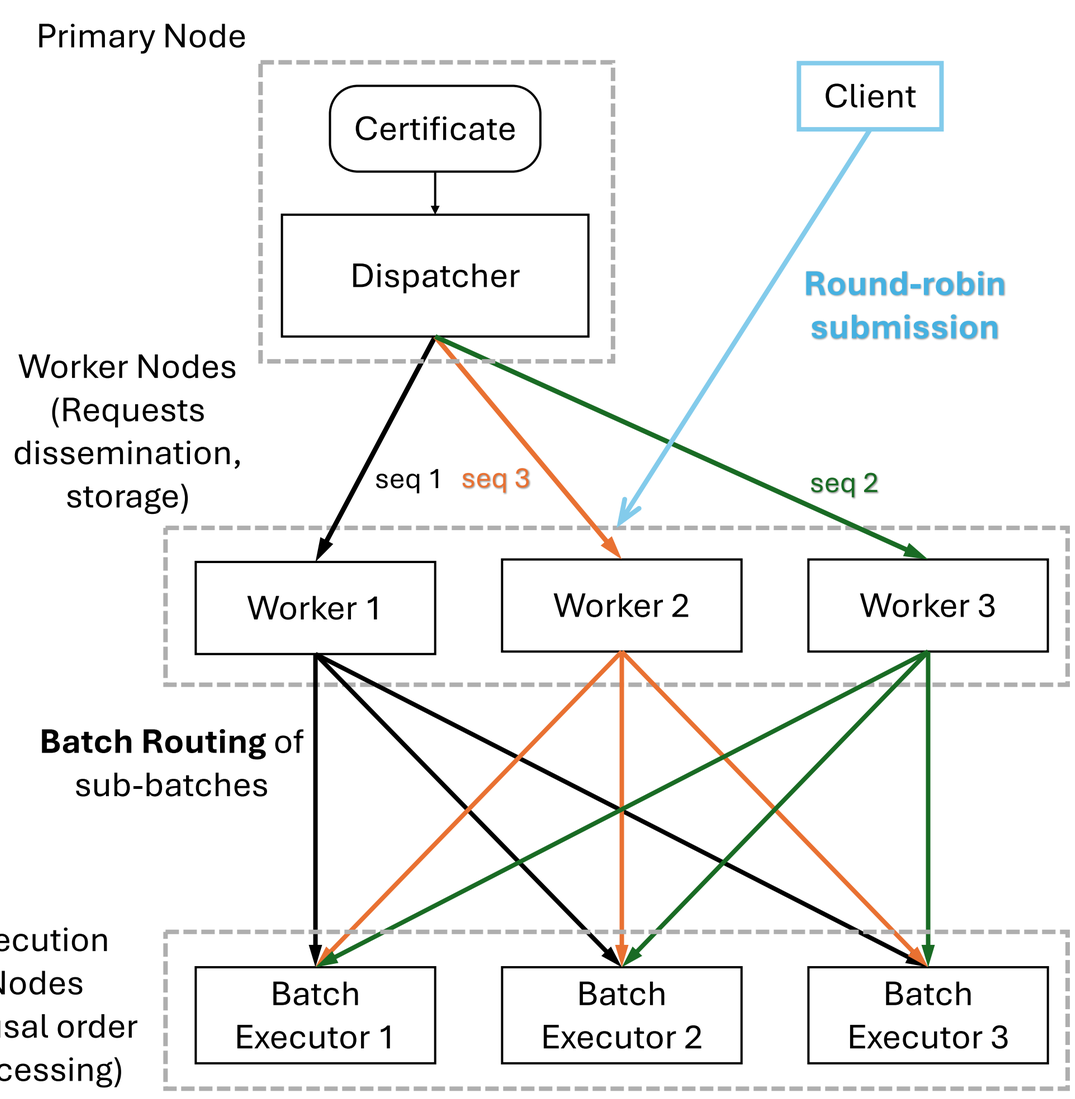
Which validator's worker should a client submit its requests to?
How do we prevent stragglers from stalling the DAG?
Does fixing the worker layer break the execution layer?

Architecture and The Execution Bottleneck

- **Batch Routing:** We introduce a Batch Router in the worker node that enables **round-robin submission**. This addresses worker-level imbalance, keeping committed throughput high.
- **Internal Routing:** Execution routing occurs internally in each validator party a batch is broken into sub-batches with sequence numbers [7] to ensure **total ordering**.
- **Distributed transactions execution without 2PC**: once the certificates are totally ordered, we rely on the deterministic ordering [4] to execute distributed transactions.

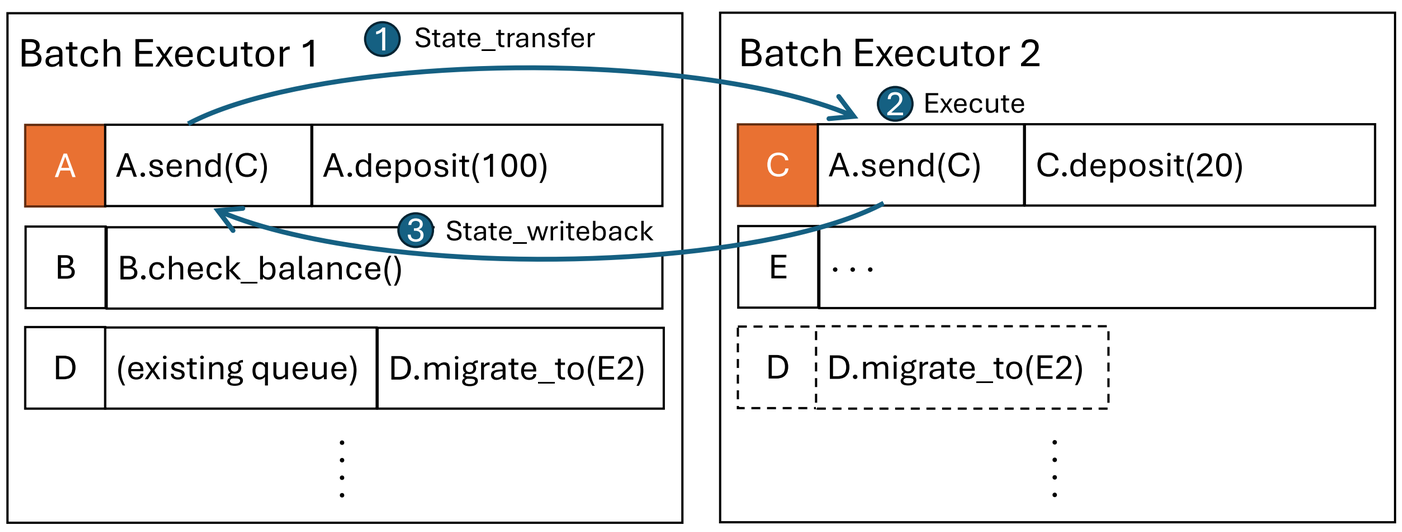


- **The Execution Bottleneck:** While committed TPS is stable, **End-to-End TPS drops by 80%** with just 10% distributed transactions. The **impact of skew shifts to the execution layer.**



Proposed Solution: Executors load balancing and state migration

- **Two-Tier Strategy**[2]**:** We use a lightweight metric (Queue Length) to trigger a heavy-duty load balancing algorithm
  - Tier 1: Monitors the imbalance ratio of the sum of queue lengths on each executor.
  - Tier 2: Utilizes Clay[5] (greedy heuristic) or Schism[6] (hypergraph partitioning) for load balancing.
- We found that weighting *queue length* higher than *execution count* is essential for vertex weighting. Distributed transactions stall the queue, so low execution count ≠ low load in concurrent execution engine.
- **Migration as a Transaction:** State transfer is treated as a **control request** ordered *within* the normal transaction stream and routed to the source and destination executor.



Distributed transactions execution without 2PC And State Migration

Ongoing Work

Zero-Downtime Migration: By allowing a control request to jump queue and implementing live migration (e.g., Squall[3], MgCrab[8]), we can enable early state transfer without pausing execution.

Future Work

Validator Balancing: Inter-node balancing with algorithms that tolerate malicious behavior, possibly incorporating Machine Learning.

Open Questions

- **Topology:** Given that star topologies are inherently un-partitionable, is it valid to assume parameterized structures like multi-clusters, or must a general-purpose system solve for the worst-case graph?
- **Throughput:** Since a client broadcast to f+1 validators significantly reduces throughput via deduplication, should we shift to optimistic submission (send to one with timeout) despite the risk of higher tail latency?
- **Workloads:** While Smallbank is the standard benchmark, is it sufficient to prove our contribution, or are real-world traces required to truly stress-test execution bottlenecks?
- **Rebalancing parameters**: Load balancing relies heavily on accurate cost parameters. Is the complexity of adaptive online tuning worth the overhead compared to static configuration, and does it offer sufficient research novelty?
- **Contribution:** Does the integration of deterministic execution, load balancing, and live migration constitute a cohesive contribution, or does it risk over-engineering the system?

Reference

1. Danezis, et al. "Narwhal and tusk: A DAG-based mempool and efficient BFT consensus." EuroSys '22.
2. Taft, et al. "E-store: Fine-grained elastic partitioning for distributed transaction processing systems" VLDB'14.
3. Elmore, et al. "Squall: Fine-grained live reconfiguration for partitioned main memory databases." SIGMOD'15.
4. Alexander, et al. "Calvin: fast distributed transactions for partitioned database systems." SIGMOD '12
5. Serafini, et al. "Clay: Fine-grained adaptive partitioning for general database schemas." VLDB'16.
6. Carlo, et al. "Schism: a workload-driven approach to database replication and partitioning" ." VLDB'10.
7. Kniep, et al. "Pilotfish: Distributed Execution for Scalable Blockchains" FC'25.
8. Lin, et al. "MgCrab: transaction crabbing for live migration in deterministic database systems" ." VLDB'19.