PEEP: A Parallel Execution Engine for Permissioned Blockchain Systems

Zhihao Chen, Xiaodong Qi, Xiaofan Du, Zhao Zhang, Cheqing Jin
chenzh@stu.ecnu.edu.cn

華東師範大學
EAST CHINA NORMAL UNIVERSITY

DaSE Data Science & Engineering
SCHOOL OF DATA SCIENCE & ENGINEERING
数据科学与工程学院

Outline

- Introduction
- Contribution
- Related Work
- System Overview
- System Details
- Experiment
- Conclusion

Introduction

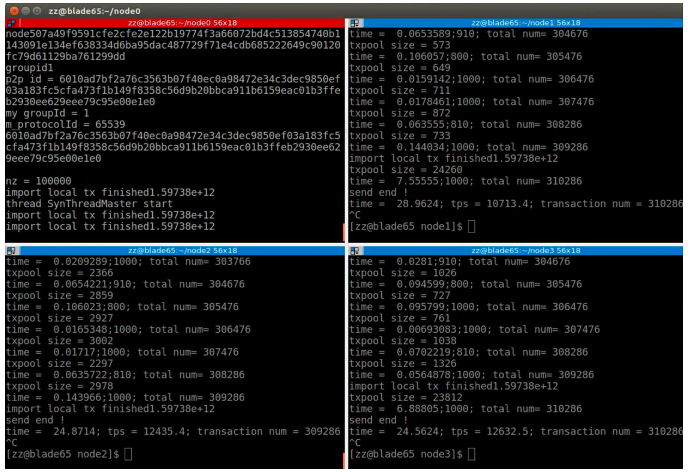
- Blockchain provides data integrity, transparency and immutability to tackle trust problems among mutually distrusting parties

- Permissioned blockchain is being widely applied to support large-scale businesses in enterprise collaborations

Introduction

- Permissioned settings opens an opportunity for seeking higher throughput under stricter assumptions

- However, there is a big performance gap between efficient consensus and conventional serial execution



- We have measured the performance of PBFT on 4 nodes in a private network, which results in over 15K TPS
- While the execution only reaches half order-of-magnitude

Contribution

- We propose **PEEP**, a novel execution engine incorporate concurrency for order-execute blockchain
- To further scale the transaction processing, we present a lock-free parallel update algorithm for state trie and realize the non-blocked workflow
- We implement a prototype for PEEP integrating above techniques and conduct extensive experiments

Related Work (Optimized execution)

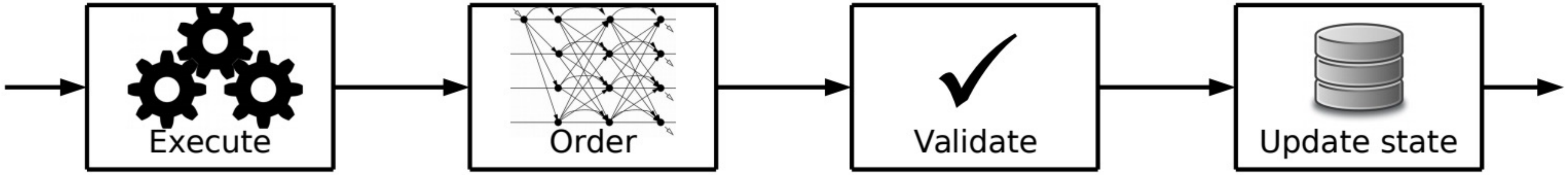
- Two-phase execution framework
  - blocked & poor performance

- Parblockchain exploits transaction parallelism (order-execute paradigm)
  - priori knowledge of transactions
  - using dependency graph

order-execute

Related Work (Optimized execution)

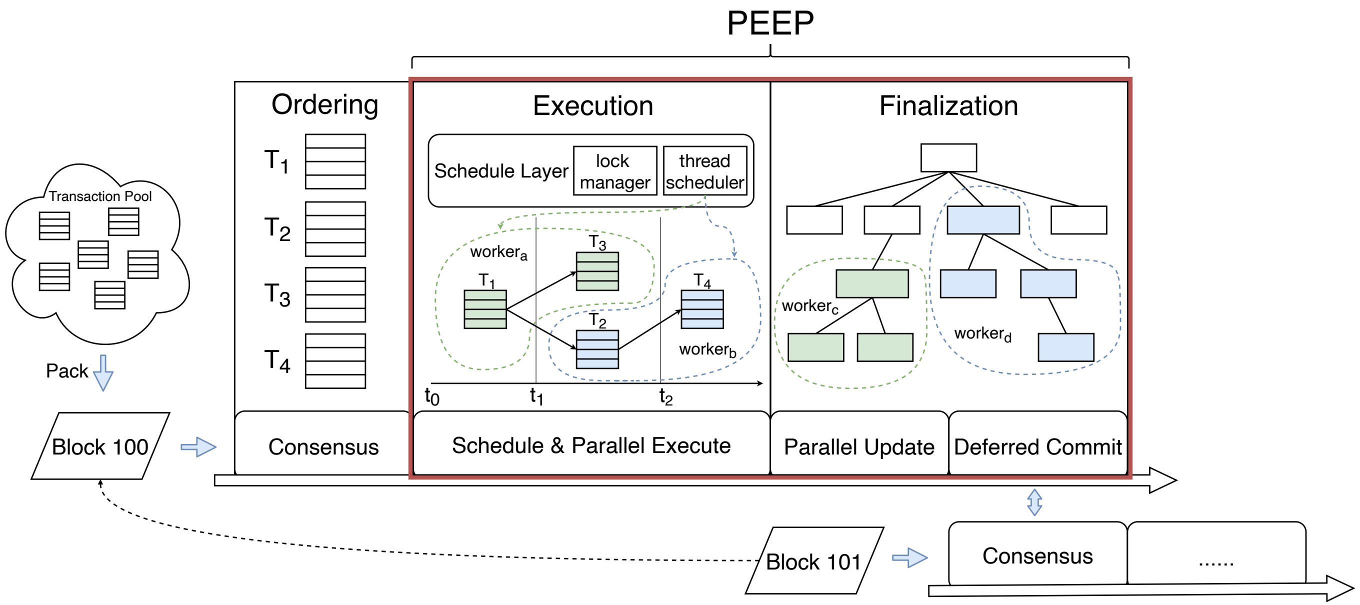
- Fabric incorporate concurrency (execute-order-validate-commit paradigm)
  - serial validation
  - high abort rates for hotspot workloads
  - enhanced works still inherits the limitations of serial validation



execute-order-validate-commit

System Overview

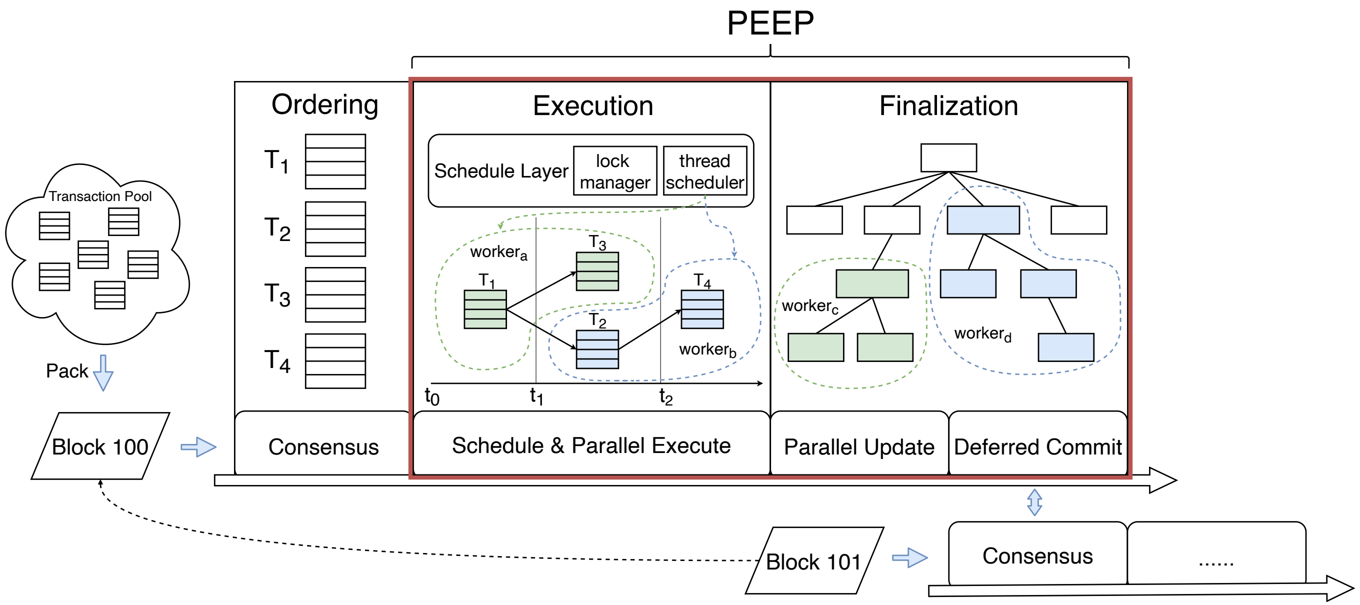
- Assumptions & Architecture
  - order-execute workflow
  - early read/write-set acquisition
  - adversary model



architecture

System Overview

- Ordering
  - merely order the transactions through consensus

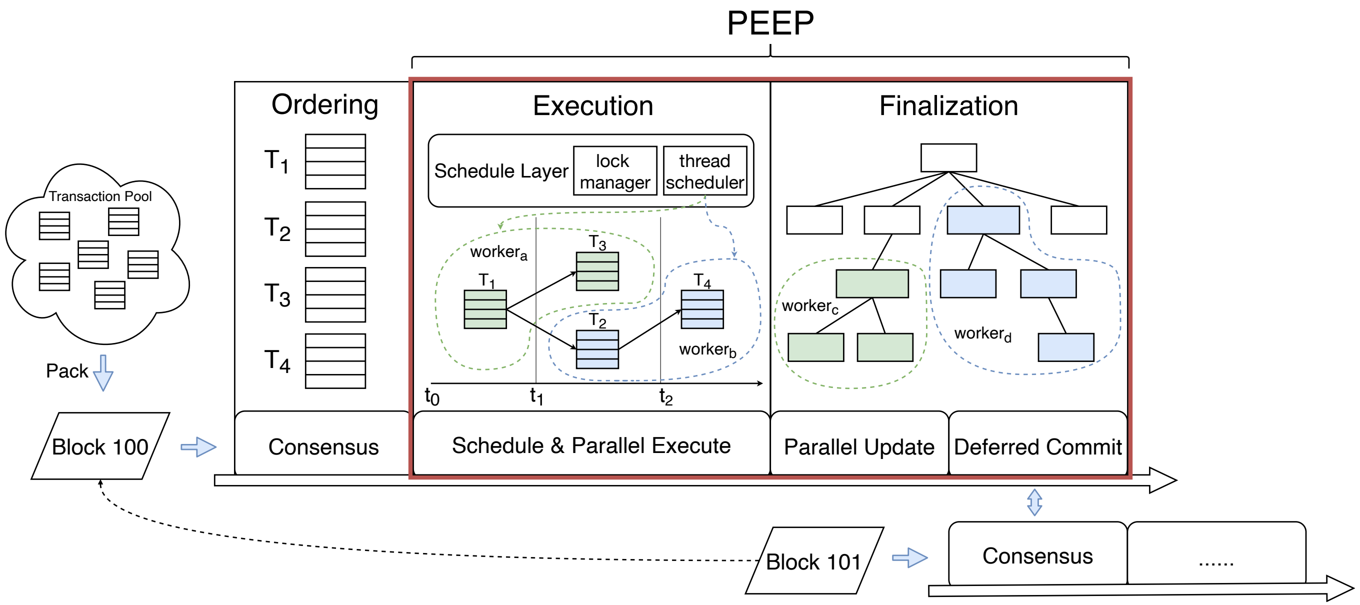


architecture

System Overview

- Execution
  - allow parallel execution with deterministic serial order
  - incur no additional network communication among replicas for negotiation of the final result

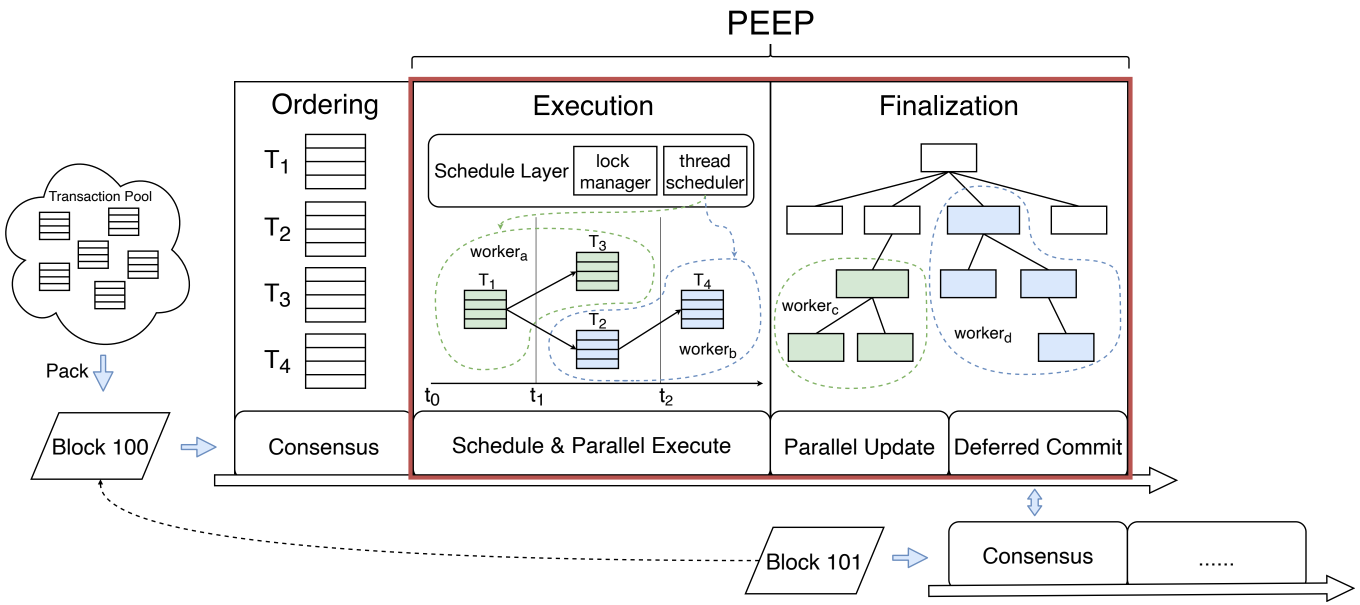
defined by ordering



architecture

System Overview

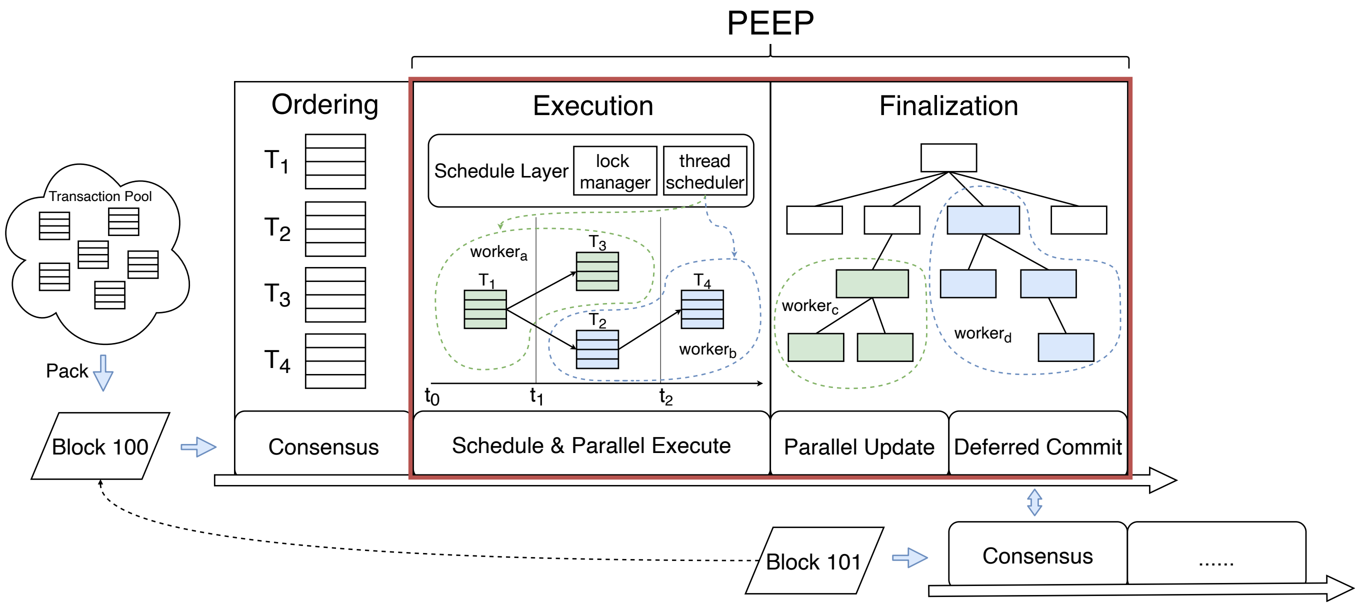
- Finalization
  - take full advantage of the batch property
  - parallelizes the updating on state trie



architecture

System Overview

- Non-Blocked Workflow
  - deferred commit strategy
  - better utilization of various resources



architecture

System Details

- Parallel execution
  - challenge: eliminate non-determinism among replicas

$A_{init}=B_{init}=0$

	T_1 A=A+2	T_2 B=B+1	T_3 A=A*2

Replica R_1: L(1,A) L(2,B) U(1,A) L(3,A) U(2,B) U(3,A)
Serial Order: T_1 T_2 T_3 Final Value: A=4, B=1

......

Replica R_4: L(2,B) L(1,A) U(2,B) U(1,A) L(3,A) U(3,A)
Serial Order: T_1 T_2 T_3 Final Value: A=4, B=1

determinism

$A_{init}=B_{init}=0$

	T_1 A=A+2	T_2 B=B+1	T_3 A=A*2

Replica R_1: L(3,A) L(2,B) U(3,A) L(1,A) U(2,B) U(1,A)
Serial Order: T_3 T_2 T_1 Final Value: A=2, B=1

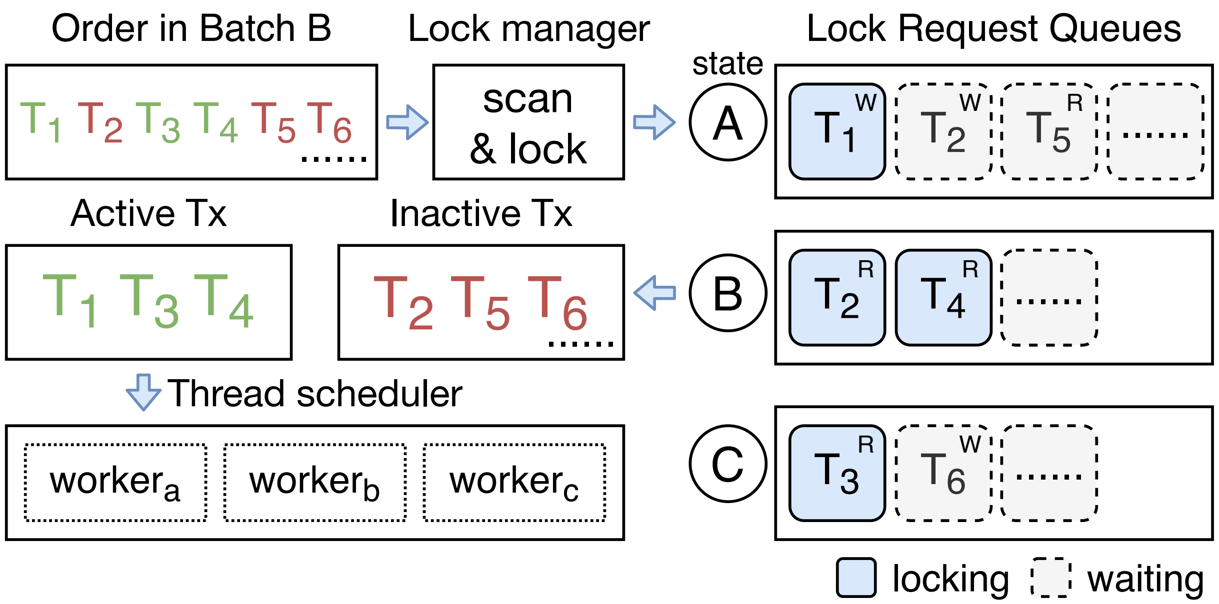
......

Replica R_4: L(1,A) L(2,B) U(2,B) U(1,A) L(3,A) U(3,A)
Serial Order: T_2 T_1 T_3 Final Value: A=4, B=1

non-determinism

System Details

- Parallel execution
  - solution: schedule transactions based on ordered locking mechanism
  - single schedule thread and multiple worker threads



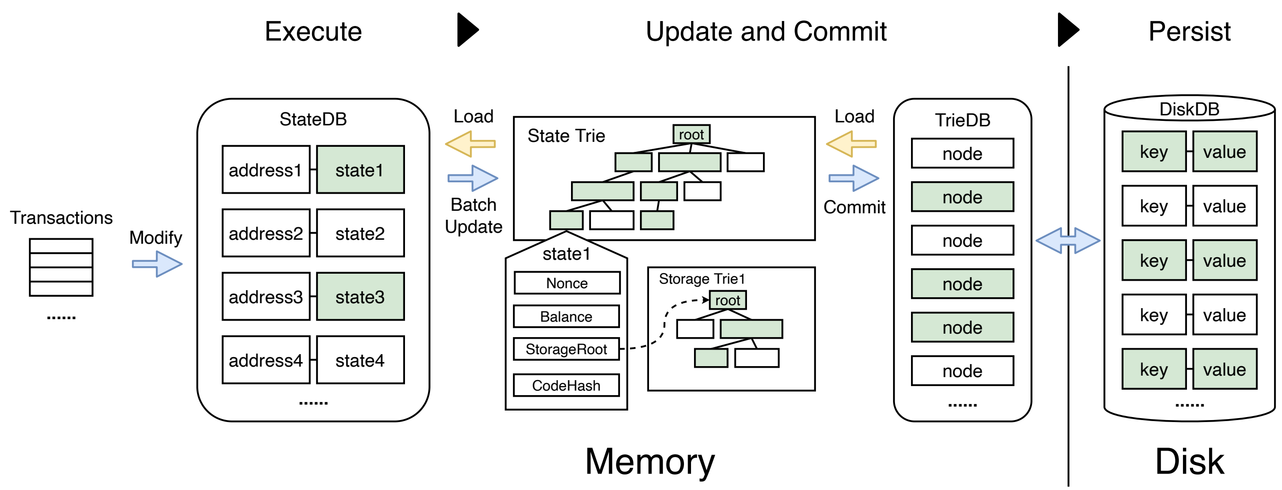
lock example of ordered locking

System Details

- Advantages & Discussion
  - improve the overall performance
  - prevent non-deterministic behaviors
  - allow each replica to execute transactions in parallel independently
  - incur non-logical aborts mentioned as inherent limitations in Fabric
  - only acquire the key in prior

System Details

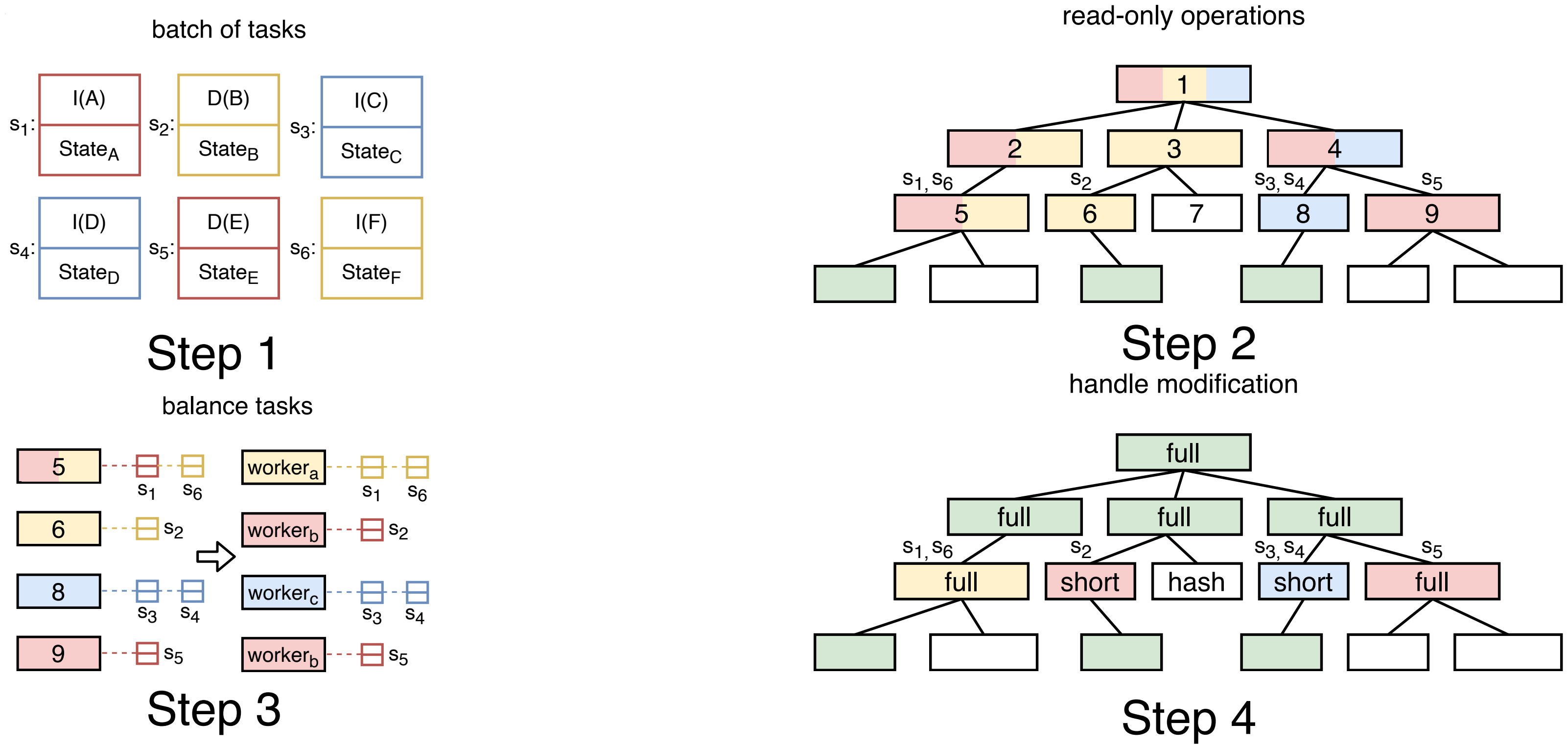
- Parallel Finalization
  - few works explore concurrency on Merkle trees
  - utilize the batch property
  - design a lock-free parallel update algorithm on state trie



The role of state trie

System Details

- Parallel Finalization
  - regard each update ⟨k,v⟩ as a task
  - detect the potential contention among tasks



System Details

- Deferred commit strategy
  - reform the state root padding
  - leading to a non-blocked workflow
  - various system resources of each replica are fully utilized, e.g., computation、disk I/O、network I/O
  - at the acceptable cost of a deferred authenticate query

Experiment

- Implementation
  - ordering:
    - we integrate an open-source PBFT realization in Hyperledger Fabric 0.6
  - execution:
    - we set an EVM(Ethereum VM) pool with multiple instances to provide the execution environment
  - finalization:
    - we reuse the MPT component in Ethereum and implement our parallel update algorithm upon it

Experiment

- Setup
  - All experiments are conducted on four qualified machines with an Intel Core i7-7700HQ CPU @ 2.80GHz of 4 cores, 16GB RAM and 1TB disk space
  - Benchmark:
    - a SmallBank contract as Macro-benchmark in BlockBench is deployed on one account
    - 1 million transactions are sent by 200 thousand accounts to invoke this smart contract under Zipfian distribution.
    - a block is limited to contain at most 1 thousand transactions

Experiment

- Overall Performance

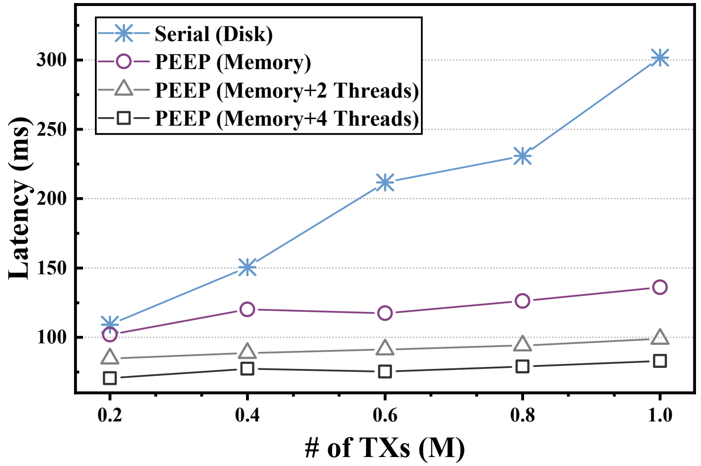


Fig1. Lantency

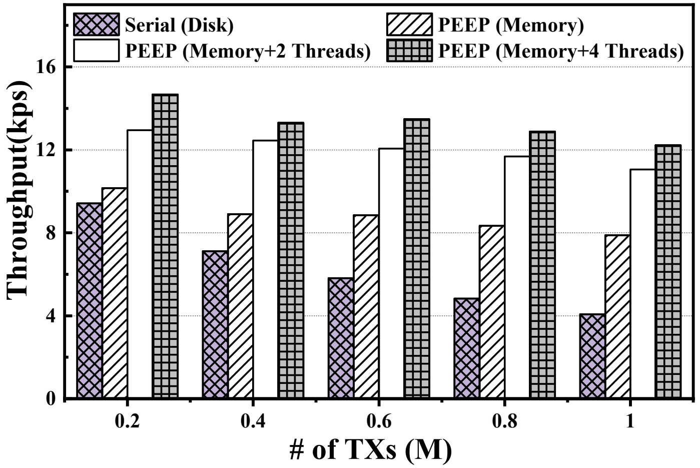


Fig2. Throughput

- against the number of transactions without consensus
- 50% overall enhancement

Experiment

- Analysis of parallel execution

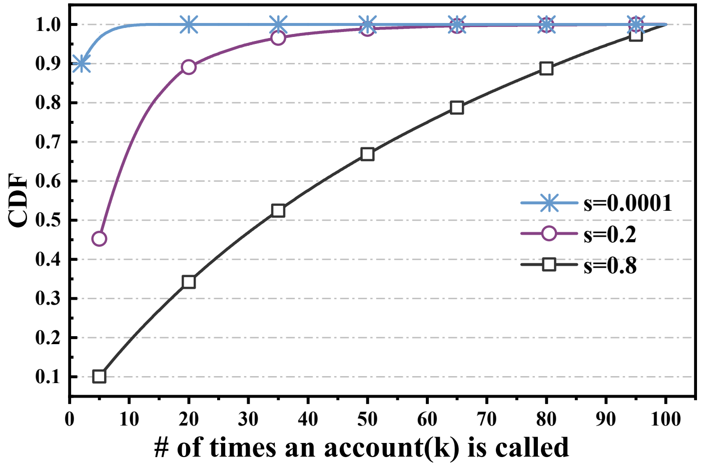


Fig3. CDF of # account calls

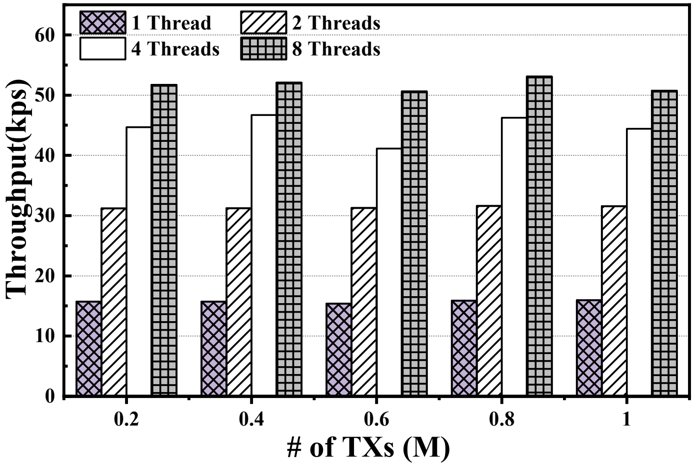


Fig4. Throughput

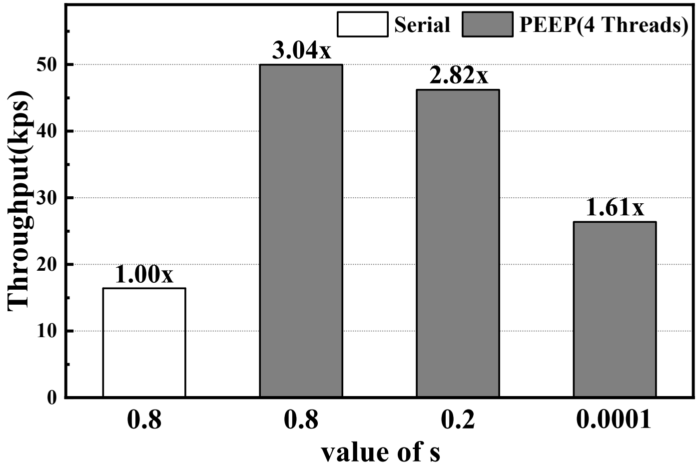


Fig5. Performance degradation

- significant improvement
- degrade under extreme contention

Experiment

- Analysis of parallel finalization

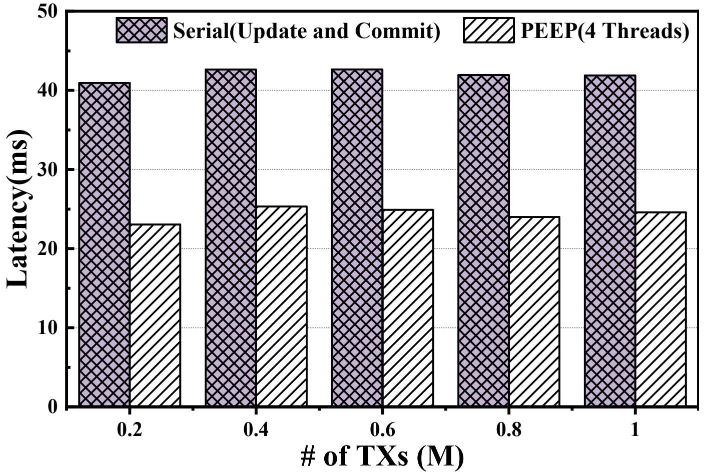


Fig6. Latency per block

  - empower the finalization by one order of magnitude

Experiment

- Analysis of deferred commit strategy

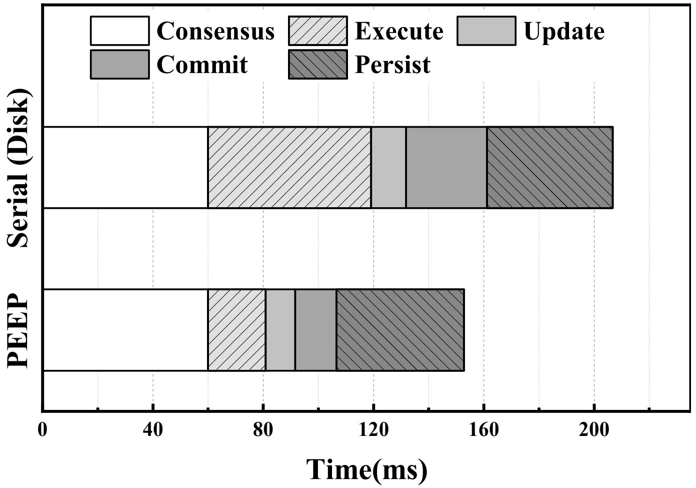


Fig7. Time of each sub-phases

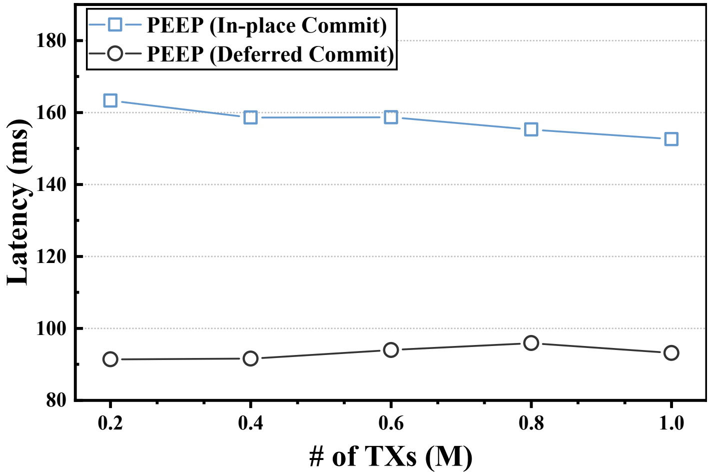


Fig8. Lantency

  - non-blocked workflow
  - early confirmation

Conclusion

- We introduces PEEP, a parallel execution engine designed for permissioned blockchain systems for higher throughput

- Optimized execution and finalization with a deferred strategy
  - experimental results support the practicability and effectiveness

- Future works
  - explore pipelined workflow
  - design more efficient authenticated data structures

THANKS !