भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

DiPETrans: A Framework for Distributed Parallel Execution of Transactions of Blocks in Blockchain*

(Annual Progress Seminar)

Parwat Singh Anjana (CS17RESCH11004)

Guided by:
Dr. Sathya Peri, Associate Professor

Department of Computer Science and Engineering,
Indian Institute of Technology Hyderabad, India

*Accepted at Concurrency and Computation: Practice and Experience (CCPE), Wiley, 2021

Outline

1. Introduction
2. Bottleneck in Existing Blockchain Design
3. Challenges in Executing Transactions Parallelly
4. Current Progress
5. Experimental Evaluation
6. Conclusion and Future Work

Outline

1. Introduction

2. Bottleneck in Existing Blockchain Design

3. Challenges in Executing Transactions Parallelly

4. Current Progress

5. Experimental Evaluation

6. Conclusion and Future Work

Introduction: Blockchain

- Blockchain is a distributed, decentralized database or ledger of records.

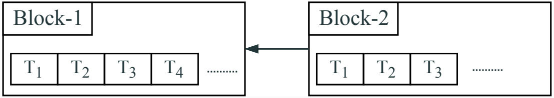
[1] https://bitcoin.org/en/

[2] https://www.ethereum.org/

[3] https://www.hyperledger.org/

Introduction: Blockchain

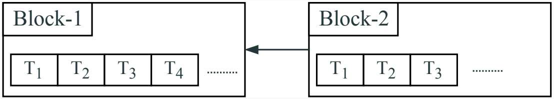
- Blockchain is a distributed, decentralized database or ledger of records.



[1] https://bitcoin.org/en/
[2] https://www.ethereum.org/
[3] https://www.hyperledger.org/

Introduction: Blockchain

- Blockchain is a distributed, decentralized database or ledger of records.

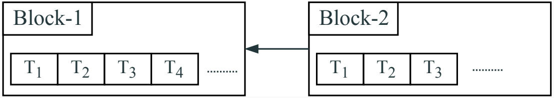


- Miners add blocks to the blockchain, and validators validate each block added to the blockchain.

[1] https://bitcoin.org/en/
[2] https://www.ethereum.org/
[3] https://www.hyperledger.org/

Introduction: Blockchain

- Blockchain is a distributed, decentralized database or ledger of records.



- Miners add blocks to the blockchain, and validators validate each block added to the blockchain.
- Example: Bitcoin[1], **Ethereum**[2], Fabric, Sawtooth[3], etc.

▸ Execution of Ethereum

[1] https://bitcoin.org/en/
[2] https://www.ethereum.org/
[3] https://www.hyperledger.org/

Outline

1. Introduction
2. Bottleneck in Existing Blockchain Design
3. Challenges in Executing Transactions Parallelly
4. Current Progress
5. Experimental Evaluation
6. Conclusion and Future Work

Bottleneck in Existing Blockchain: Ethereum

- Serial execution of the transactions by miners and validators fails to harness the power of multi-core processors', thus degrading throughput.

Bottleneck in Existing Blockchain: Ethereum

- Serial execution of the transactions by miners and validators fails to harness the power of multi-core processors', thus degrading throughput.

- By leveraging multiple threads to execute transactions, we can achieve better efficiency and higher throughput.

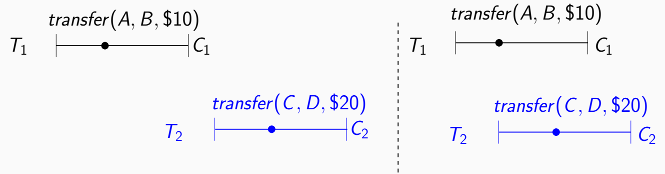
Bottleneck in Existing Blockchain: Ethereum

- Serial execution of the transactions by miners and validators fails to harness the power of multi-core processors', thus degrading throughput.

- By leveraging multiple threads to execute transactions, we can achieve better efficiency and higher throughput.

Listing 1: Transfer function

```
transfer(s_id, r_id, amt) {
  if(amt > bal[s_id])
    throw;
  bal[s_id] -= amt;
  bal[r_id] += amt;
}
```



(a) Serial Execution of transactions (b) Concurrent Execution

Figure 1: Motivation towards concurrent execution over serial

Outline

1. Introduction
2. Bottleneck in Existing Blockchain Design
3. Challenges in Executing Transactions Parallelly
4. Current Progress
5. Experimental Evaluation
6. Conclusion and Future Work

Parallel Execution Challenges (1/4)

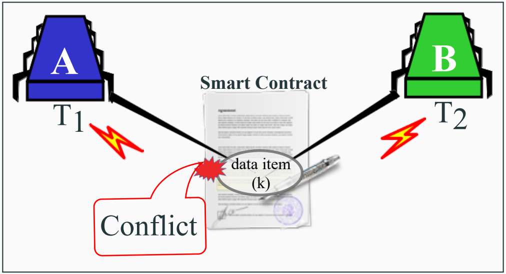


Figure 2: Conflicting access to shared data item.

Parallel Execution Challenges (1/4)

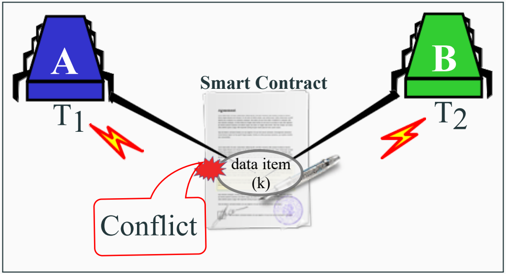


Figure 2: Conflicting access to shared data item.

- Identifying the conflicts at run-time is not straightforward.

Parallel Execution Challenges (1/4)

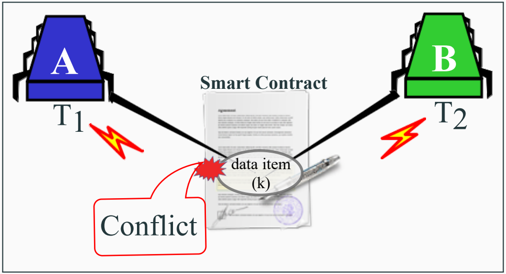


Figure 2: Conflicting access to shared data item.

- Identifying the conflicts at run-time is not straightforward.
- Improper use of locks may lead to deadlock.

Parallel Execution Challenges (1/4)

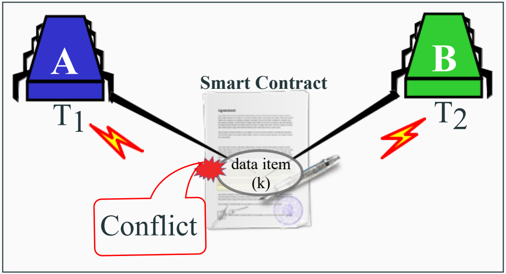


Figure 2: Conflicting access to shared data item.

- Identifying the conflicts at run-time is not straightforward.
- Improper use of locks may lead to deadlock.
- Discovering an equivalent serial schedule of concurrent execution of SCTs is difficult.

Parallel Execution Challenges (1/4)

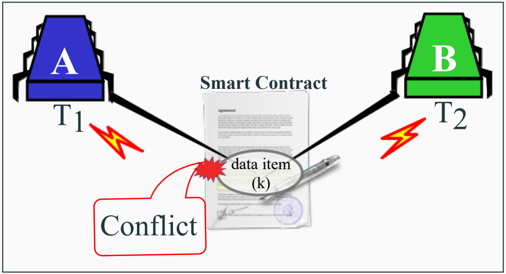


Figure 2: Conflicting access to shared data item.

- Identifying the conflicts at run-time is not straightforward.
- Improper use of locks may lead to deadlock.
- Discovering an equivalent serial schedule of concurrent execution of SCTs is difficult.

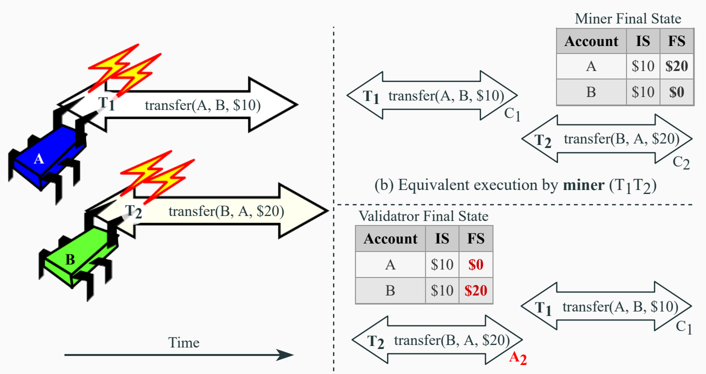
Solution: We use *Software Transactional Memory Systems (STMs)* to solve these challenges.

Parallel Execution Challenges (2/4)

- Validator may incorrectly reject a valid block proposed by the miner. We call such error the **False Block Rejection (FBR)** error.

Parallel Execution Challenges (2/4)

- Validator may incorrectly reject a valid block proposed by the miner. We call such error the **False Block Rejection (FBR)** error.



(a) Concurrent execution

(b) Equivalent execution by **miner** (T_1T_2)

(c) Equivalent execution by **validator** (T_2T_1)

Parallel Execution Challenges (3/4)

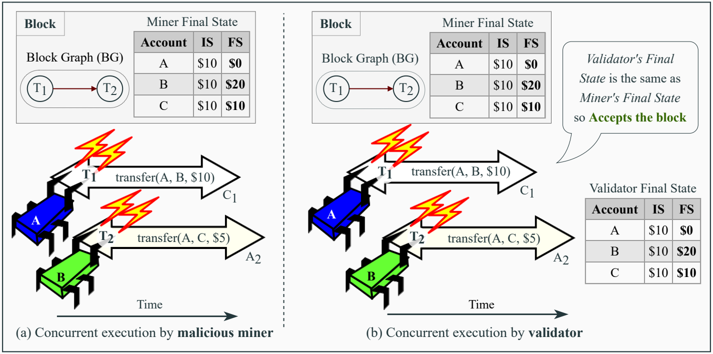
Solution: Miner appends the *Block Graph (BG)*[4,5] in the block to avoid the FBR error.

[4] Dickerson et al., *"Adding Concurrency to Smart Contracts."* PODC, 2017

[5] Anjana et al., *"An efficient framework for optimistic concurrent execution of smart contracts."* PDP, 2019

Parallel Execution Challenges (3/4)

Solution: Miner appends the *Block Graph (BG)*[4,5] in the block to avoid the FBR error.



4 Dickerson et al., *"Adding Concurrency to Smart Contracts."* PODC, 2017

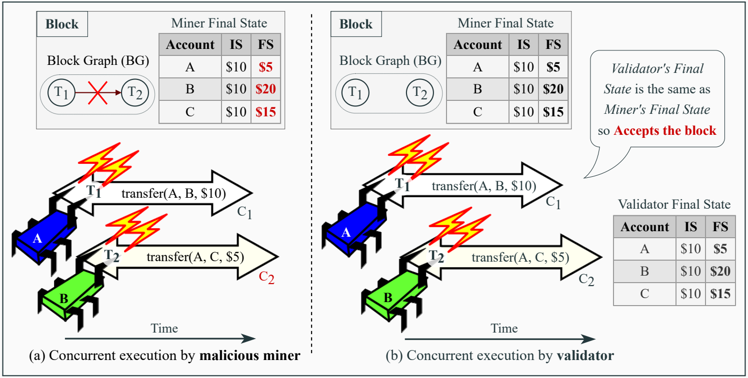
5 Anjana et al., *"An efficient framework for optimistic concurrent execution of smart contracts."* PDP, 2019

Parallel Execution Challenges (4/4)

- A *Malicious miner* can send an incorrect Block Graph to harm the blockchain, missing some edges, e.g., to cause *double spending*. We call such error the **Edge Missing BG (EMB)** error.

Parallel Execution Challenges (4/4)

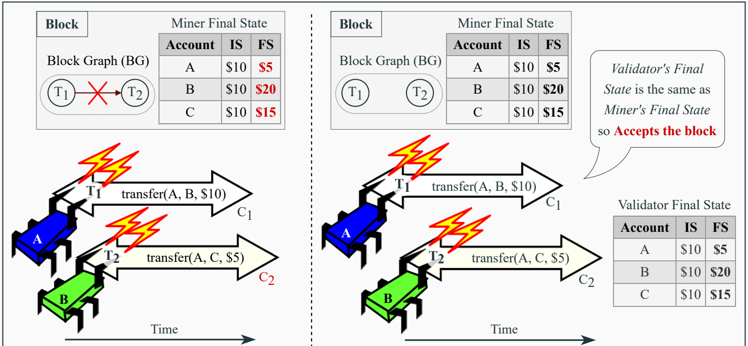
- A *Malicious miner* can send an incorrect Block Graph to harm the blockchain, missing some edges, e.g., to cause *double spending*. We call such error the **Edge Missing BG (EMB)** error.



(a) Concurrent execution by **malicious miner**
(b) Concurrent execution by **validator**

Parallel Execution Challenges (4/4)

- A *Malicious miner* can send an incorrect Block Graph to harm the blockchain, missing some edges, e.g., to cause *double spending*. We call such error the **Edge Missing BG (EMB)** error.



(a) Concurrent execution by **malicious miner** (b) Concurrent execution by **validator**

Solution: We propose a *Smart Multi-threaded Validator (SMV)* to detect EMB error and rejects the corresponding blocks.

Outline

1. Introduction
2. Bottleneck in Existing Blockchain Design
3. Challenges in Executing Transactions Parallelly
4. Current Progress
5. Experimental Evaluation
6. Conclusion and Future Work

Proposed Approach: DiPETrans Framework

- We proposed a *DiPETrans* framework[6] for parallel execution of the transactions at miners and validators, based on transaction shards identified using static analysis.

[6] *Accepted at the journal of **"Concurrency and Computation: Practice and Experience (CCPE),"** Wiley, 2021.

Proposed Approach: DiPETrans Framework

- We proposed a *DiPETrans* framework[6] for parallel execution of the transactions at miners and validators, based on transaction shards identified using static analysis.

[6] *Accepted at the journal of **"Concurrency and Computation: Practice and Experience (CCPE),"** Wiley, 2021.

Proposed Approach: DiPETrans Framework

- We proposed a *DiPETrans* framework[6] for parallel execution of the transactions at miners and validators, based on transaction shards identified using static analysis.
- We implement this technique using a distributed *leader–follower* approach within a mining community of servers.

[6] * Accepted at the journal of **"Concurrency and Computation: Practice and Experience (CCPE),"** Wiley, 2021.

Proposed Approach: DiPETrans Framework

- We proposed a *DiPETrans* framework[6] for parallel execution of the transactions at miners and validators, based on transaction shards identified using static analysis.
- We implement this technique using a distributed *leader–follower* approach within a mining community of servers.
- The leader shards the transactions in the block and the followers concurrently execute (mining) or verify (validation) them.

[6] *Accepted at the journal of **"Concurrency and Computation: Practice and Experience (CCPE),"** Wiley, 2021.

Proposed Approach: DiPETrans Framework

- We proposed a *DiPETrans* framework[6] for parallel execution of the transactions at miners and validators, based on transaction shards identified using static analysis.
- We implement this technique using a distributed *leader–follower* approach within a mining community of servers.
- The leader shards the transactions in the block and the followers concurrently execute (mining) or verify (validation) them.
- When mining, the PoW is also partitioned and solved in parallel by the members of the community.

[6] *Accepted at the journal of **"Concurrency and Computation: Practice and Experience (CCPE),"** Wiley, 2021.

Proposed Approach: Sharding of Block Transactions

- DiPETrans groups the block transactions into independent shards and executes them parallelly in a distributed fashion using a leader-follower approach.

Transaction	AccountsAccessed	
T_1 (A_1, A_3)	A_1	A_3
T_2 (A_9, A_0)	A_9	A_0
T_3 (A_0, A_9)	A_0	A_9
T_4 (A_4, A_6)	A_4	A_6
T_5 (A_1, A_8)	A_1	A_8
T_6 (A_6, A_7)	A_6	A_7
T_7 (A_2, A_3)	A_2	A_3
T_8 (A_4)	A_4	
T_9 (A_5, A_9)	A_5	A_9

→ Dependencies

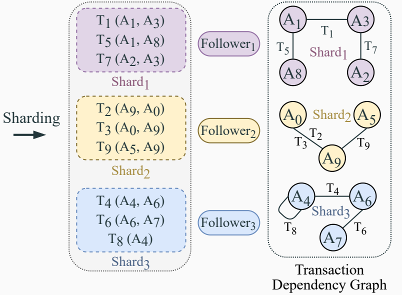
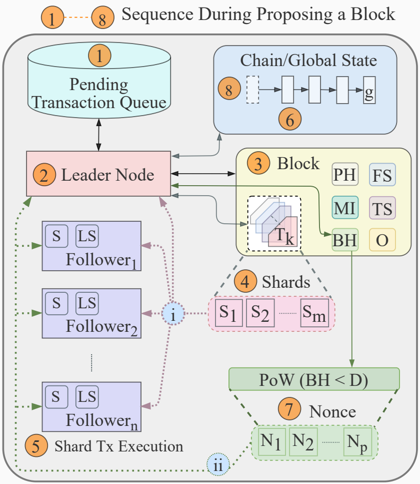


Figure 3: Sharding of transactions in a block using static graph analysis

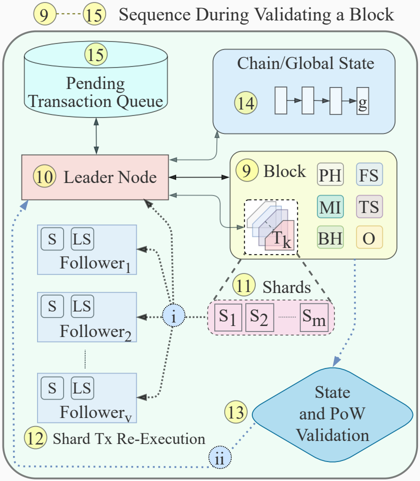
DiPETrans Architecture: Miner Community



(a) Community Acting as Miner

MI: Miner ID
TS: Timestamp
D: Difficulty
BH: Block Hash
PH: Previous Hash
Tx: Transaction
FS: Final State
LS: Local State
O: Other Information
S: Shard
m: # Shards
k: # Transactions
p: # Nonce Set
g: Genesis Block
n: # Followers at Miner
v: # Followers at Validator

DiPETrans Architecture: Validator Community



(b) Community Acting as Validator

MI: Miner ID
TS: Timestamp
D: Difficulty
BH: Block Hash
PH: Previous Hash
Tx: Transaction
FS: Final State
LS: Local State
O: Other Information
S: Shard
m: # Shards
k: # Transactions
p: # Nonce Set
g: Genesis Block
n: # Followers at Miner
v: # Followers at Validator

DiPETrans: Theoretical Running Time Complexity

- `Analyze()` takes $\mathcal{O}(n)$ to build transaction graph with n edges and between 2 - $2n$ vertices. So, static analysis using WCC takes $\mathcal{O}(n)$.

[7] The time to complete the transaction execution is limited by the follower with the most number of transactions.

DiPETrans: Theoretical Running Time Complexity

- `Analyze()` takes $\mathcal{O}(n)$ to build transaction graph with n edges and between 2 - $2n$ vertices. So, static analysis using WCC takes $\mathcal{O}(n)$.
- With m shards and f follower nodes in the community. The `LoadBalance()` takes $\mathcal{O}(m \cdot log(m))$ to sort the shards.

[7] The time to complete the transaction execution is limited by the follower with the most number of transactions.

DiPETrans: Theoretical Running Time Complexity

- `Analyze()` takes $\mathcal{O}(n)$ to build transaction graph with n edges and between 2 - $2n$ vertices. So, static analysis using WCC takes $\mathcal{O}(n)$.
- With m shards and f follower nodes in the community. The `LoadBalance()` takes $\mathcal{O}(m \cdot log(m))$ to sort the shards.
- Using a priority queue to load balance shards (transactions) assigned to each follower, we get a time complexity of $\mathcal{O}(m \cdot log(f))$.

[7] The time to complete the transaction execution is limited by the follower with the most number of transactions.

DiPETrans: Theoretical Running Time Complexity

- `Analyze()` takes $\mathcal{O}(n)$ to build transaction graph with n edges and between 2 - $2n$ vertices. So, static analysis using WCC takes $\mathcal{O}(n)$.
- With m shards and f follower nodes in the community. The `LoadBalance()` takes $\mathcal{O}(m \cdot log(m))$ to sort the shards.
- Using a priority queue to load balance shards (transactions) assigned to each follower, we get a time complexity of $\mathcal{O}(m \cdot log(f))$.
- For the `LoadBalance` phase the combined time complexity is $\mathcal{O}(m \cdot (log(m) + log(f)))$.

[7] The time to complete the transaction execution is limited by the follower with the most number of transactions.

DiPETrans: Theoretical Running Time Complexity

- `Analyze()` takes $\mathcal{O}(n)$ to build transaction graph with n edges and between 2 - $2n$ vertices. So, static analysis using WCC takes $\mathcal{O}(n)$.
- With m shards and f follower nodes in the community. The `LoadBalance()` takes $\mathcal{O}(m \cdot log(m))$ to sort the shards.
- Using a priority queue to load balance shards (transactions) assigned to each follower, we get a time complexity of $\mathcal{O}(m \cdot log(f))$.
- For the `LoadBalance` phase the combined time complexity is $\mathcal{O}(m \cdot (log(m) + log(f)))$.
- So overall time complexity of $\mathcal{O}(n + m \cdot (log(m) + log(f)))$. Usually, with $m > f$, expected complexity is $\mathcal{O}(n + m \cdot log(m))$.

[7] The time to complete the transaction execution is limited by the follower with the most number of transactions.

DiPETrans: Theoretical Running Time Complexity

- `Analyze()` takes $\mathcal{O}(n)$ to build transaction graph with n edges and between 2 - $2n$ vertices. So, static analysis using WCC takes $\mathcal{O}(n)$.
- With m shards and f follower nodes in the community. The `LoadBalance()` takes $\mathcal{O}(m \cdot log(m))$ to sort the shards.
- Using a priority queue to load balance shards (transactions) assigned to each follower, we get a time complexity of $\mathcal{O}(m \cdot log(f))$.
- For the `LoadBalance` phase the combined time complexity is $\mathcal{O}(m \cdot (log(m) + log(f)))$.
- So overall time complexity of $\mathcal{O}(n + m \cdot (log(m) + log(f)))$. Usually, with $m > f$, expected complexity is $\mathcal{O}(n + m \cdot log(m))$.
- The worst-case time complexity for transaction execution is $\mathcal{O}(n \cdot t_x)$ and the best-case time complexity is $\Omega(\frac{n}{f} \cdot t_x)$, where, t_x is a transaction execution time.[7]

[7] The time to complete the transaction execution is limited by the follower with the most number of transactions.

Outline

1. Introduction
2. Bottleneck in Existing Blockchain Design
3. Challenges in Executing Transactions Parallelly
4. Current Progress
5. Experimental Evaluation
6. Conclusion and Future Work

DiPETrans: Experimental Evaluation

- We empirically evaluated DiPETrans using 5 million actual transactions from the Ethereum blockchain.

DiPETrans: Experimental Evaluation

- We empirically evaluated DiPETrans using 5 million actual transactions from the Ethereum blockchain.
  - We extracted $\approx$ 80K blocks consisting of 5,170,597 total transactions.

DiPETrans: Experimental Evaluation

- We empirically evaluated DiPETrans using 5 million actual transactions from the Ethereum blockchain.
  - We extracted $\approx$ 80K blocks consisting of 5,170,597 total transactions.
  - There are two types of transactions: *monetary* and *smart contracts*.

DiPETrans: Experimental Evaluation

- We empirically evaluated DiPETrans using 5 million actual transactions from the Ethereum blockchain.
  - We extracted $\approx$ 80K blocks consisting of 5,170,597 total transactions.
  - There are two types of transactions: *monetary* and *smart contracts*.
- We used a commodity cluster to run the leader and followers.

DiPETrans: Experimental Evaluation

- We empirically evaluated DiPETrans using 5 million actual transactions from the Ethereum blockchain.
  - We extracted $\approx$ 80K blocks consisting of 5,170,597 total transactions.
  - There are two types of transactions: *monetary* and *smart contracts*.
- We used a commodity cluster to run the leader and followers.
  - The implementation is in *C++* using Apache thrift cross-platform micro-services library.

DiPETrans: Experimental Evaluation

- We empirically evaluated DiPETrans using 5 million actual transactions from the Ethereum blockchain.
    - We extracted $\approx$ 80K blocks consisting of 5,170,597 total transactions.
    - There are two types of transactions: *monetary* and *smart contracts*.
- We used a commodity cluster to run the leader and followers.
    - The implementation is in *C++* using Apache thrift cross-platform micro-services library.
    - Each node in the cluster has an 8-core AMD CPU with 32 GB memory, running CentOS, and connected using 1 Gbps Ethernet.

DiPETrans: Experimental Evaluation

- We empirically evaluated DiPETrans using 5 million actual transactions from the Ethereum blockchain.
  - We extracted $\approx$ 80K blocks consisting of 5,170,597 total transactions.
  - There are two types of transactions: *monetary* and *smart contracts*.
- We used a commodity cluster to run the leader and followers.
  - The implementation is in *C++* using Apache thrift cross-platform micro-services library.
  - Each node in the cluster has an 8-core AMD CPU with 32 GB memory, running CentOS, and connected using 1 Gbps Ethernet.
- Depending on the experiment configuration, a community has a leader running on one node and between 1 to 5 followers running on separate nodes.

DiPETrans: Experiment Workload

Table 1: Summary of transactions in experiment workload

Block type	ρ	# Txns/ block	# Blocks	$\sum$# Contract txns	$\sum$# Non-contract txns
data-1-1-100		100	3,880	193,959	194,000
data-1-1-200		200	1,940	193,959	194,000
data-1-1-300	$\frac{1}{1}$	300	1,294	193,959	194,100
data-1-1-400		400	970	193,959	194,000
data-1-1-500		500	776	193,959	194,000
data-1-2-100		100	5,705	193,959	376,530
data-1-2-200		200	2,895	193,959	385,035
data-1-2-300	$\frac{1}{2}$	300	1,940	193,959	388,000
data-1-2-400		400	1,448	193,959	385,168
data-1-2-500		500	1,162	193,959	386,946
data-1-4-100		100	9,698	193,959	775,840
data-1-4-200		200	4,849	193,959	775,840
data-1-4-300	$\frac{1}{4}$	300	3,233	193,959	775,840
data-1-4-400		400	2,425	193,959	776,000
data-1-4-500		500	1,940	193,959	776,000
data-1-8-100		100	16,164	193,959	1,422,432
data-1-8-200		200	8,434	193,959	1,492,818
data-1-8-300	$\frac{1}{8}$	300	5,705	193,959	1,517,530
data-1-8-400		400	4,311	193,959	1,530,405
data-1-8-500		500	3,464	193,959	1,538,016
data-1-16-100		100	32,327	193,959	3,038,738
data-1-16-200		200	16,164	193,959	3,038,832
data-1-16-300	$\frac{1}{16}$	300	10,776	193,959	3,038,832
data-1-16-400		400	8,082	193,959	3,038,832
data-1-16-500		500	6,466	193,959	3,039,020

DiPETrans Results: Transaction Execution Speedup on W1

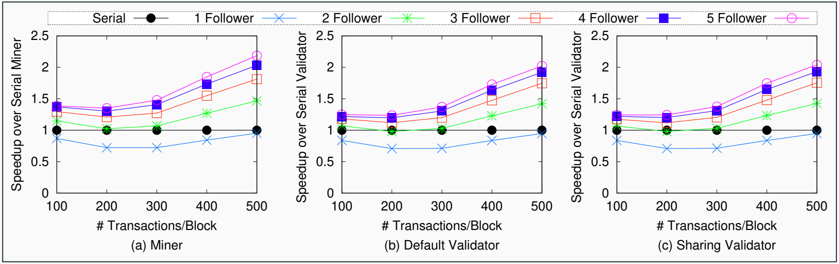


Figure 4: Workload-1: speedup by community miner and validator over serial miner and validator.

- With 5 followers, the peak speedup achieved by the community miners' is 2.18×, the speedup efficiency is sub-optimal at about 51% for 4 followers and 44% for 5 followers, with 500 transactions/blocks.
- The default community validators' average speedup is 1.25×, and their peak is 2.03× with 5 followers and 500 transactions per block.

DiPETrans Results: Transaction Execution Speedup on W2

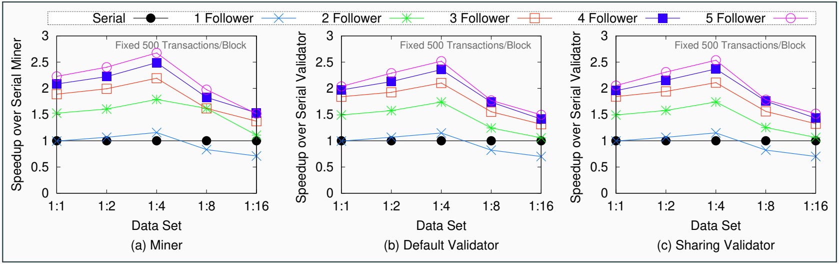


Figure 5: Workload-2: speedup by community miner and validator over serial miner and validator.

- For the community miners' a peak speedup of 2.7× is achieved with 5 followers and a favorable speedup efficiency of 73% with 3 followers is achieved when $\rho = \frac{1}{4}$.
- For the default community validators' a peak speedup of 2.5× is achieved with 5 followers.

DiPETrans Results: End-to-end Mining Speedup

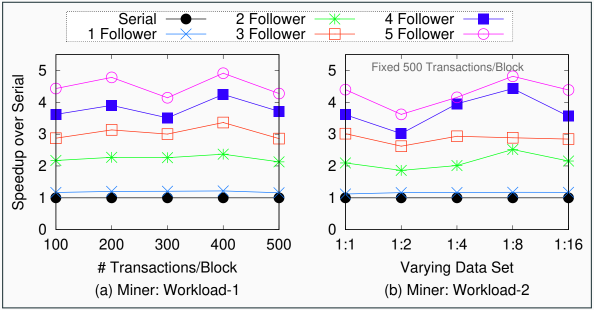


Figure 6: Average end-to-end block creation speedup by community miner over serial miner.

- In Workload 1, a speedup of 1.15× to 4.91× for 1–5 followers that remain stable as the block size increases, with a speedup efficiency of 57.5 to 81.83%.
- We achieve a maximum speedup of 1.17× to 4.82× for 1–5 followers, with a speedup efficiency of 58.5 to 80.33% in Workload 2.

DiPETrans Results: Throughput

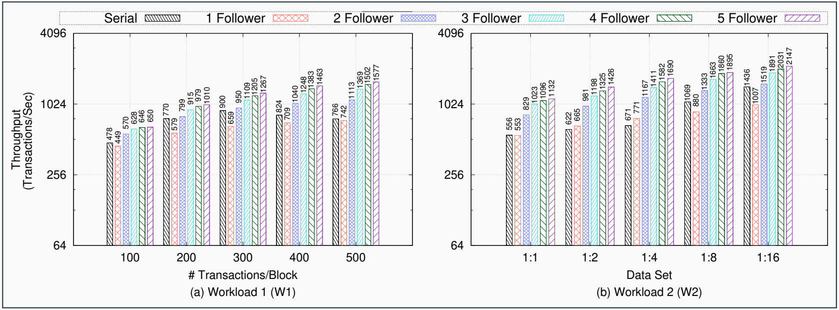


Figure 7: Throughput with varying transactions per block and varying ρ.

- In Workload 1, the maximum throughput is 1577 tps in a community with 5 followers at 500 transactions/block, which is 2.05× higher than that of serial execution.
- In Workload 2, we achieves a maximum throughput of 2147 tps that is 1.49× over serial when ratio $\rho = \frac{1}{16}$ for 5 followers, with 500 transactions/blocks. The sweet spot of maximum throughput is 2.52× with 1690 tps when $\rho = \frac{1}{4}$.

DiPETrans Results: Optimal Community Size

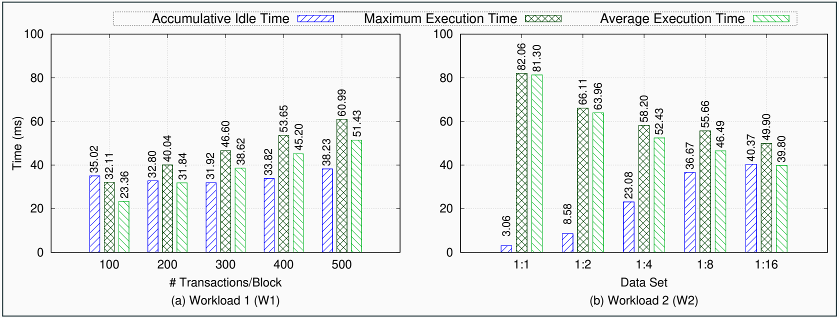


Figure 8: Transaction execution time by a follower and accumulative followers idle time on W1 and W2.

- The optimal community size depends on several parameters: # transactions/block, # shards formed, the mix of contractual and monetary transactions/shard.
- With an optimal community size, the idle time will be minimized, hence, the average execution time will be similar to the maximum execution time.

Outline

1. Introduction
2. Bottleneck in Existing Blockchain Design
3. Challenges in Executing Transactions Parallelly
4. Current Progress
5. Experimental Evaluation
6. Conclusion and Future Work

DiPETrans Conclusion

- We proposed DiPETrans framework to execute block transactions efficiently in parallel by leveraging distributed resources using leader-follower approach.

DiPETrans Conclusion

- We proposed DiPETrans framework to execute block transactions efficiently in parallel by leveraging distributed resources using leader-follower approach.
- The proposed techniques prevent transaction parallelization errors such as FBR, EMB, and FBin.

DiPETrans Conclusion

- We proposed DiPETrans framework to execute block transactions efficiently in parallel by leveraging distributed resources using leader-follower approach.
- The proposed techniques prevent transaction parallelization errors such as FBR, EMB, and FBin.
- We achieve a maximum speedup of $2.2\times$ and $2.0\times$ and an average speedup of $1.6\times$ and $1.5\times$ for the miner and the validator, respectively, with 100 to 500 transactions per block when using 6 machines in the community.

Future Work

- Exploring the possibilities of integrating our ideas into existing order-execute-based blockchain platforms like Bitcoin, Sawtooth, Tezos, and EOS is an exciting direction to pursue.

Future Work

- Exploring the possibilities of integrating our ideas into existing order-execute-based blockchain platforms like Bitcoin, Sawtooth, Tezos, and EOS is an exciting direction to pursue.
- We plan to integrate it with Ethereum blockchain by deploying a *DiPETrans* community smart contract.

Future Work

- Exploring the possibilities of integrating our ideas into existing order-execute-based blockchain platforms like Bitcoin, Sawtooth, Tezos, and EOS is an exciting direction to pursue.
- We plan to integrate it with Ethereum blockchain by deploying a *DiPETrans* community smart contract.
- Another interesting direction is to apply concurrency in the nested execution of SCTs.

Collaborators

Sathya Peri
Associate Professor
IIT Hyderabad, India
sathya_p@cse.iith.ac.in

Yogesh Simmhan
Professor
IISc, Bangalore, India
simmhan@iisc.ac.in

Parwat Singh Anjana
Ph.D. Student
IIT Hyderabad, India
cs17resch11004@iith.ac.in

Shrey Baheti
Software Engineer
Cargill Digital Labs, India
shrey_baheti@cargill.com

Thanks!

Publications (1/2)

Journal Papers:

- Shrey Baheti, **Parwat Singh Anjana**, Sathya Peri, and Yogesh Simmhan. *"DiPETrans: A Framework for Distributed Parallel Execution of Transactions of Blocks in Blockchain."* **CCPE**, Wiley, (In Press, Accepted on Nov. 28, 2021).
- **Parwat Singh Anjana**, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. *"OptSmart: A Space Efficient Optimistic Concurrent Execution of Smart Contracts."* **SI on Blockchain, DAPD**, Springer (Under Revision), 2021.

Conference Papers:

- **Parwat Singh Anjana**, Hagit Attiya, Sweta Kumari, Sathya Peri, and Archit Somani. *"Efficient Concurrent Execution of Smart Contracts in Blockchains using Object-based Transactional Memory."* **NETYS**, pp. 77 - 93, Springer, 2021.
- **Parwat Singh Anjana**, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. *"An Efficient Framework for Optimistic Concurrent Execution of Smart Contracts."* **PDP**, pp. 83 - 92, IEEE, 2019.

Short Papers:

- **Parwat Singh Anjana**. *"Efficient Parallel Execution of Block Transactions in Blockchain."* **Middleware Doctoral Symposium**, pp. 8 - 11, ACM, 2021.
- Prashansa Agrawal, **Parwat Singh Anjana**, and Sathya Peri. *"DeHiDe: Deep Learning-based Hybrid Model to Detect Fake News using Blockchain."* **ICDCN**, pp. 245 – 246, ACM, 2021.
- **Parwat Singh Anjana**, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. *"Entitling concurrency to smart contracts using optimistic transactional memory."* **ICDCN**, pp. 508 - 508, ACM, 2019. (**Best Poster Award**)

Publications (2/2)

Manuscripts under review/preparation:

- **Parwat Singh Anjana**, Adithya Rajesh Chandrassery, and Sathya Peri. *"An Efficient Approach to Move Elements in the Distributed Geo-Replicated Tree."* **CCGrid**, Under Review, 2022.
- **Parwat Singh Anjana**, Shailesh Mishra, and Sathya Peri. *"BDIDS: A Blockchain-based Distributed Intrusion Detection System for IoT Networks."* Manuscript Under Preparation, 2022.
- **Parwat Singh Anjana**, Sai Ramana Reddy, and Sathya Peri. *"Empirical Study of Parallel Execution of Block Transactions in the Tezos and Ethereum Blockchain."* Manuscript Under Preparation, 2022.
- **Parwat Singh Anjana**, Sandeep Kulkarni, Sathya Peri, Raaghav Ravishankar, and Diksha Sethi. *"Caliber-GC: A Causally Consistent Space Efficient Geo-Replicated Distributed Key-value Store."* Manuscript Under Preparation, 2022.

Introduction: Ethereum High Level Design

- Ethereum nodes form a peer-to-peer system.
- Clients (external to the system) wishing to execute smart contracts, contact a peer of the system.

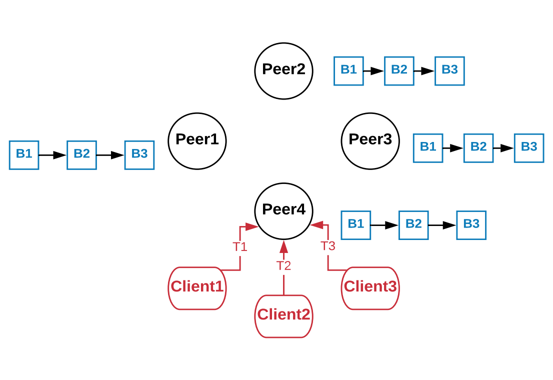


Figure 9: Clients send Transaction T1, T2 and T3 to Miner (Peer4)

Introduction: Ethereum High Level Design

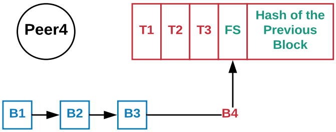


Figure 10: Miner forms a block B4 and computes final state (FS) sequentially

Introduction: Ethereum High Level Design

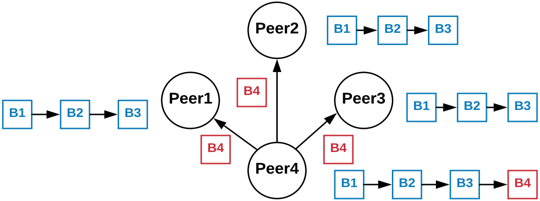


Figure 11: Miner broadcasts the block B4

Introduction: Ethereum High Level Design

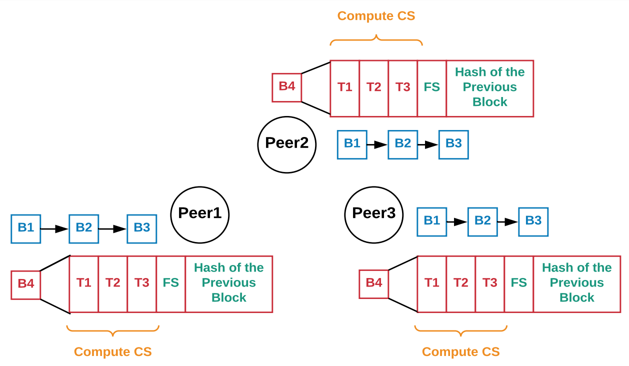


Figure 12: Validators (Peer 1, 2, and 3) compute current state (CS) sequentially

Introduction: Ethereum High Level Design

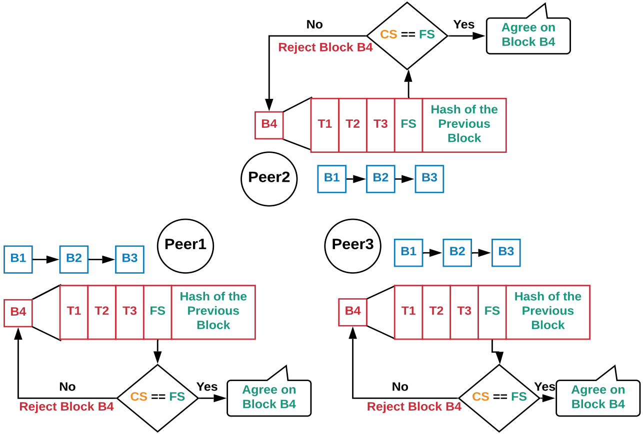


Figure 13: Validators verify the FS and reach the consensus protocol

Introduction: Ethereum High Level Design

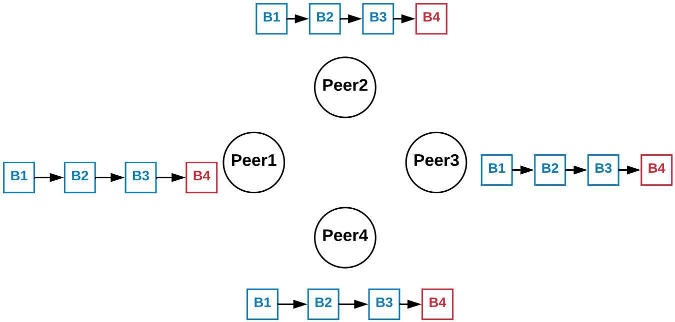


Figure 14: Block B4 successfully added to the blockchain

IITH-STM Library

- We have used two protocols implemented in IITH-STM library for concurrent execution of the smart contracts by miner.

  1. Basic Time-stamp Ordering (BTO) Protocol.
  2. Multi-Version Time-stamp Ordering (MVTO) Protocol.

Basic Time-stamp Ordering (BTO) Protocol[8]

- If $p_i(x)$ and $q_j(x)$, i $\neq$ j, are operations in conflict, the following has to hold:
  - $p_i(x)$ is executed before $q_j(x)$ iff $ts(t_i) < ts(t_j)$.

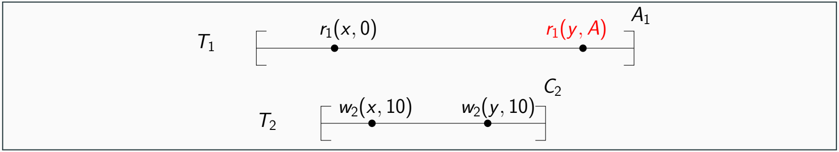


Figure 15: BTO

[8] Gerhard Weikum and Gottfried Vossen. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery, 2002.

Multi-Version Time-stamp Ordering (MVTO) Protocol[9]

- MVTO maintains multiple versions corresponding to each shared data-objects.
- It reduces the number of aborts and improves the throughput.

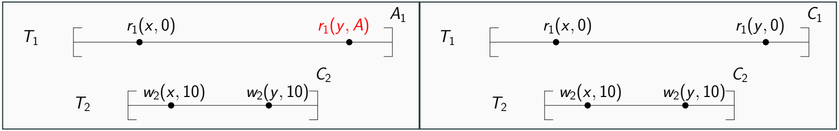


Figure 16: BTO

Figure 17: MVTO

[9] Kumar et al. A TimeStamp Based Multi-version STM Algorithm. In ICDCN, 2014

Concurrent Validator: Fork-Join Approach

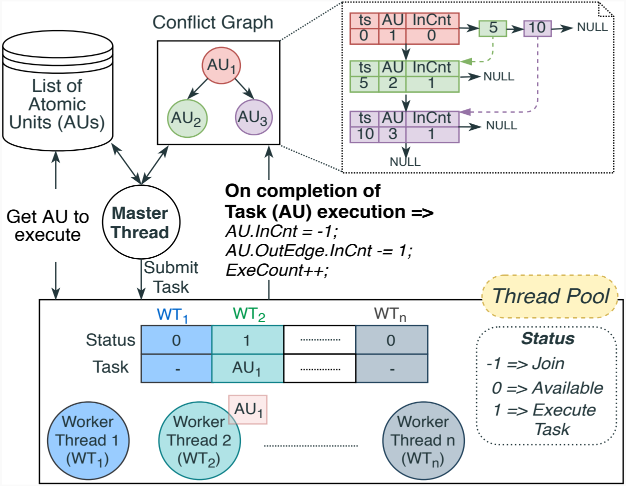


Figure 18: Fork-Join Approach

Concurrent Validator: Decentralized Approach

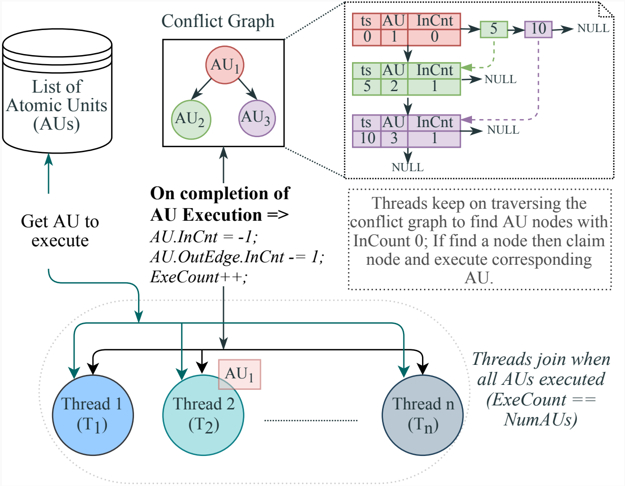


Figure 19: Decentralized Approach

Proposed Methodologies: OptSmart

- Since static analysis fails to identify the conflicts precisely.

Proposed Methodologies: OptSmart

- Since static analysis fails to identify the conflicts precisely.
- We introduce *OptSmart: A Space Efficient Optimistic Concurrent Execution of Smart Contracts* to exploit multi-processing on a multi-core system to improve throughput.

Proposed Methodologies: OptSmart

- Since static analysis fails to identify the conflicts precisely.
- We introduce *OptSmart: A Space Efficient Optimistic Concurrent Execution of Smart Contracts* to exploit multi-processing on a multi-core system to improve throughput.
- Miners and validators use multiple threads to parallelly execute smart contract transactions (SCTs) in a block.

Proposed Methodologies: OptSmart

- Since static analysis fails to identify the conflicts precisely.
- We introduce *OptSmart: A Space Efficient Optimistic Concurrent Execution of Smart Contracts* to exploit multi-processing on a multi-core system to improve throughput.
- Miners and validators use multiple threads to parallelly execute smart contract transactions (SCTs) in a block.
- A miner concurrently executes SCTs using optimistic read-write software transactional memory systems (RWSTMs) and saves the non-conflicting SCTs in the concurrent bin and conflicting SCTs in the block graph (BG).

Proposed Methodologies: OptSmart

- Since static analysis fails to identify the conflicts precisely.
- We introduce *OptSmart: A Space Efficient Optimistic Concurrent Execution of Smart Contracts* to exploit multi-processing on a multi-core system to improve throughput.
- Miners and validators use multiple threads to parallelly execute smart contract transactions (SCTs) in a block.
- A miner concurrently executes SCTs using optimistic read-write software transactional memory systems (RWSTMs) and saves the non-conflicting SCTs in the concurrent bin and conflicting SCTs in the block graph (BG).
- Later, decentralized validators re-execute SCTs deterministically in parallel to validate the block by using information appended by the concurrent miner.

OptSmart Results

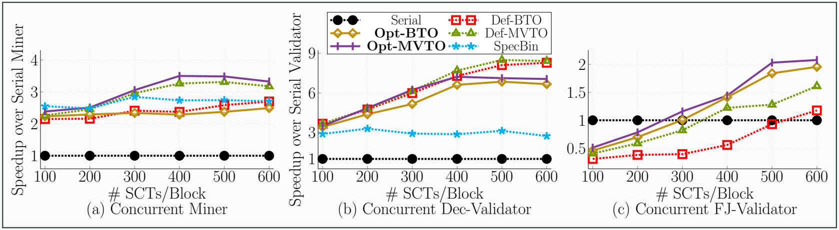


Figure 20: Speedup achieved by optimized concurrent miner and validator over serial miner and validator.

- OptSmart achieves an average speedup of 4.49× and 5.21× for **optimized concurrent miners** using **BTO (Opt-BTO)** and **MVTO STM (Opt-MVTO) protocol** than a serial miner.
- **Optimized decentralized BTO** and **MVTO concurrent validator** outperform average 7.68× and 8.60× than serial validator.
- The proposed **efficient BG** saves an average of 2.29× block space over existing approaches.

Read-Write STM (RWSTM) v/s Object-based STM (OSTM)

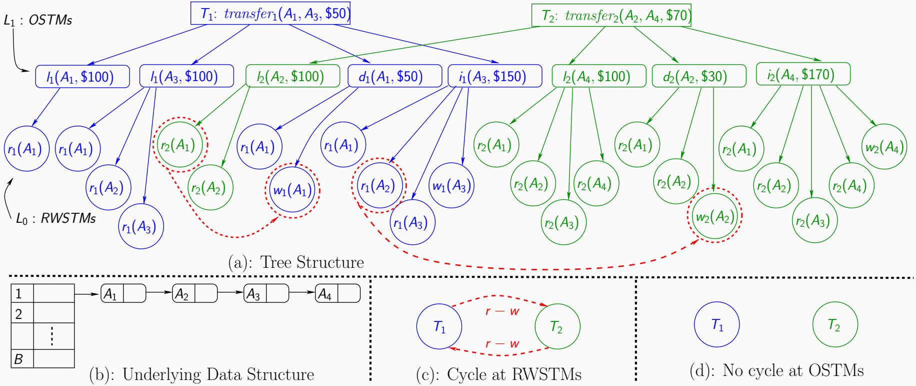


Figure 21: (a) Two SCTs T_1 and T_2 in the form of a tree structure which is working on a hash-table with B buckets where four accounts (shared data items) A_1, A_2, A_3 and A_4 are stored in the form of a list depicted in (b). T_1 transfers \$50 from A_1 to A_3 and T_2 transfers \$70 from A_2 to A_4. After checking the sufficient balance using lookup (l), SCT T_1 deletes (d) \$50 from A_1 and inserts (i) it to A_3 at higher-level (L_1). At lower-level 0 (L_0), these operations involve read (r) and write (w) to both accounts A_1 and A_3. Since, its conflict graph has a cycle either T_1 or T_2 has to abort (see (c)); However, execution at L_1 depicts that both transactions are working on different accounts and the higher-level methods are isolated. So, we can prune this tree and isolate the transactions at higher-level with equivalent serial schedule T_1T_2 or T_2T_1 as shown in (d).

Proposed Methodology: ObjSC

- We develop an efficient framework for the concurrent execution of SCTs by miners using an optimistic *Object-Based STMs (OSTMs).*[10]

[10] Peri, S., Singh, A., Somani, A.: Efficient means of Achieving Composability using Transactional Memory. NETYS, 2018.

Proposed Methodology: ObjSC

- We develop an efficient framework for the concurrent execution of SCTs by miners using an optimistic *Object-Based STMs (OSTMs).*[10]
- STMs are convenient programming paradigms for a programmer to access shared memory using multiple threads.

[10] Peri, S., Singh, A., Somani, A.: Efficient means of Achieving Composability using Transactional Memory. NETYS, 2018.

Proposed Methodology: ObjSC

- We develop an efficient framework for the concurrent execution of SCTs by miners using an optimistic *Object-Based STMs (OSTMs)*.[10]
- STMs are convenient programming paradigms for a programmer to access shared memory using multiple threads.
- Traditional STMs work on read-write primitives. We refer to these as *Read-Write STMs (RWSTMs)*.

[10] Peri, S., Singh, A., Somani, A.: Efficient means of Achieving Composability using Transactional Memory. NETYS, 2018.

Proposed Methodology: ObjSC

- We develop an efficient framework for the concurrent execution of SCTs by miners using an optimistic *Object-Based STMs (OSTMs)*.[10]
- STMs are convenient programming paradigms for a programmer to access shared memory using multiple threads.
- Traditional STMs work on read-write primitives. We refer to these as *Read-Write STMs (RWSTMs)*.
- *OSTMs* operate on higher level objects rather than primitive reads and writes which act upon memory locations.

[10] Peri, S., Singh, A., Somani, A.: Efficient means of Achieving Composability using Transactional Memory. NETYS, 2018.

Proposed Methodology: ObjSC

- We develop an efficient framework for the concurrent execution of SCTs by miners using an optimistic *Object-Based STMs (OSTMs)*.[10]
- STMs are convenient programming paradigms for a programmer to access shared memory using multiple threads.
- Traditional STMs work on read-write primitives. We refer to these as *Read-Write STMs (RWSTMs)*.
- *OSTMs* operate on higher level objects rather than primitive reads and writes which act upon memory locations.
- OSTMs provide greater concurrency than RWSTMs.

[10] Peri, S., Singh, A., Somani, A.: Efficient means of Achieving Composability using Transactional Memory. NETYS, 2018.

Proposed Methodology: ObjSC

- We develop an efficient framework for the concurrent execution of SCTs by miners using an optimistic *Object-Based STMs (OSTMs).*[10]
- STMs are convenient programming paradigms for a programmer to access shared memory using multiple threads.
- Traditional STMs work on read-write primitives. We refer to these as *Read-Write STMs (RWSTMs)*.
- *OSTMs* operate on higher level objects rather than primitive reads and writes which act upon memory locations.
- OSTMs provide greater concurrency than RWSTMs.
- Hash Table based OSTMs export the following methods:
  - STM_begin()
  - STM_insert()
  - STM_delete()
  - STM_lookup()
  - STM_tryC()
  - STM_Abort()

[10] Peri, S., Singh, A., Somani, A.: Efficient means of Achieving Composability using Transactional Memory. NETYS, 2018.

ObjSC: Thread Safe Integration of STMs in Smart Contracts

Listing 1: Transfer function

```
transfer(s_id, r_id, amt) {
 if(amt > bal[s_id])
 throw;
 bal[s_id] -= amt;
 bal[r_id] += amt;
}
```

ObjSC: Thread Safe Integration of STMs in Smart Contracts

Listing 1: Transfer function

```
1   transfer(s_id, r_id, amt) {
2    if(amt > bal[s_id])
3    throw;
4    bal[s_id] -= amt;
5    bal[r_id] += amt;
6   }
```

Listing 2: Transfer function using STM

```
7   transfer(s_id, r_id, amt) {
8    t_id = STM_begin();
9    s_bal = STM_lookup(s_id);
10   if(amt > s_bal) {
11    abort(t_id);
12    throw;
13   }
14   STM_delete(s_id, amt);
15   STM_insert(r_id, amt);
16   if(STM_tryC(t_id)!= SUCCESS)
17   goto Line 8;//Trans aborted
18  }
```

ObjSC: Working of multi-threaded miner

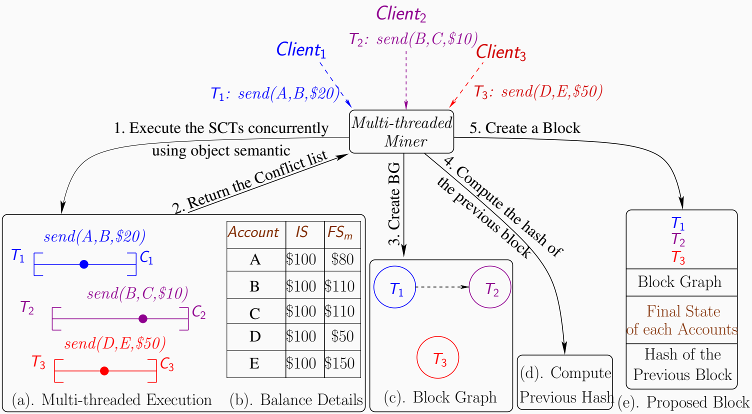


Figure 22: Working of multi-threaded miner

ObjSC: Block Graph (1/2)

- Miner maintains the BG in the form of the adjacency list, where vertices correspond only to committed SCTs.

ObjSC: Block Graph (1/2)

- Miner maintains the BG in the form of the adjacency list, where vertices correspond only to committed SCTs.
- Edges of the BG depends on the conflicts given by the OSTMs.

ObjSC: Block Graph (1/2)

- Miner maintains the BG in the form of the adjacency list, where vertices correspond only to committed SCTs.
- Edges of the BG depends on the conflicts given by the OSTMs.

$$Conflicting\ Operations = \begin{cases} STM\_lookup_i() & - \quad STM\_tryC_j() \\ STM\_delete_i() & - \quad STM\_tryC_j() \\ STM\_tryC_i() & - \quad STM\_tryC_j() \\ STM\_tryC_i() & - \quad STM\_delete_j() \\ STM\_tryC_i() & - \quad STM\_lookup_j() \end{cases} \tag{1}$$

ObjSC: Block Graph (1/2)

- Miner maintains the BG in the form of the adjacency list, where vertices correspond only to committed SCTs.
- Edges of the BG depends on the conflicts given by the OSTMs.

$$Conflicting\ \ Operations = \begin{cases} STM\_lookup_i() & - \quad STM\_tryC_j() \\ STM\_delete_i() & - \quad STM\_tryC_j() \\ STM\_tryC_i() & - \quad STM\_tryC_j() \\ STM\_tryC_i() & - \quad STM\_delete_j() \\ STM\_tryC_i() & - \quad STM\_lookup_j() \end{cases} \tag{1}$$

- **Multi-threaded miner** uses addVert() and addEdge() methods of BG.

ObjSC: Block Graph (1/2)

- Miner maintains the BG in the form of the adjacency list, where vertices correspond only to committed SCTs.
- Edges of the BG depends on the conflicts given by the OSTMs.

$$Conflicting\ Operations = \begin{cases} STM\_lookup_i() & - \quad STM\_tryC_j() \\ STM\_delete_i() & - \quad STM\_tryC_j() \\ STM\_tryC_i() & - \quad STM\_tryC_j() \\ STM\_tryC_i() & - \quad STM\_delete_j() \\ STM\_tryC_i() & - \quad STM\_lookup_j() \end{cases} \tag{1}$$

- **Multi-threaded miner** uses addVert() and addEdge() methods of BG.
- Later, validators re-execute the same SCTs concurrently and deterministically relying on the BG.

ObjSC: Block Graph (1/2)

- Miner maintains the BG in the form of the adjacency list, where vertices correspond only to committed SCTs.
- Edges of the BG depends on the conflicts given by the OSTMs.

$$Conflicting\ Operations = \begin{cases} STM\_lookup_i() & - & STM\_tryC_j() \\ STM\_delete_i() & - & STM\_tryC_j() \\ STM\_tryC_i() & - & STM\_tryC_j() \\ STM\_tryC_i() & - & STM\_delete_j() \\ STM\_tryC_i() & - & STM\_lookup_j() \end{cases} \qquad (1)$$

- **Multi-threaded miner** uses addVert() and addEdge() methods of BG.
- Later, validators re-execute the same SCTs concurrently and deterministically relying on the BG.
- Two SCTs that do not have a path can execute concurrently.

ObjSC: Block Graph (2/2)

- **SMV** uses searchGlobal() and decInCount() methods of BG.

[11] Herlihy, M., Koskinen, E.: Transactional Boosting: A Methodology for Highly-concurrent Transactional Objects. PPoPP, 2008.

ObjSC: Block Graph (2/2)

- **SMV** uses searchGlobal() and decInCount() methods of BG.

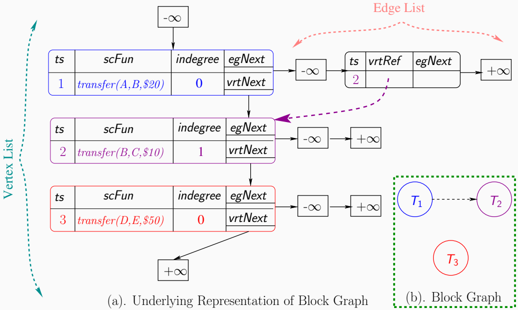


Figure 23: Data structure of BG

[11] Herlihy, M., Koskinen, E.: Transactional Boosting: A Methodology for Highly-concurrent Transactional Objects. PPoPP, 2008.

ObjSC: Block Graph (2/2)

- **SMV** uses searchGlobal() and decInCount() methods of BG.

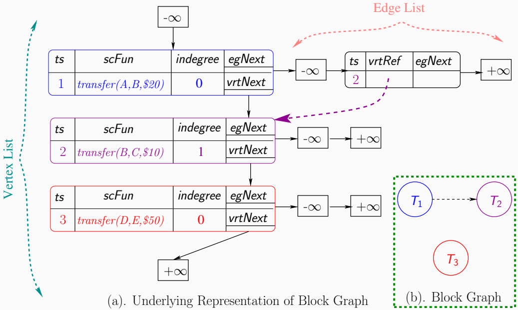


Figure 23: Data structure of BG

- OSTMs[11] have fewer conflicts than RWSTMs which in turn, allows validators to execute more SCTs concurrently.

---

[11] Herlihy, M., Koskinen, E.: Transactional Boosting: A Methodology for Highly-concurrent Transactional Objects. PPoPP, 2008.

ObjSC: Block Graph (2/2)

- **SMV** uses searchGlobal() and decInCount() methods of BG.

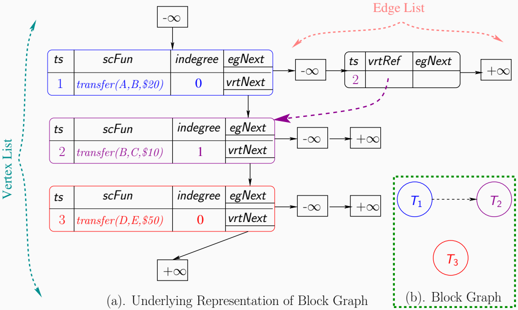
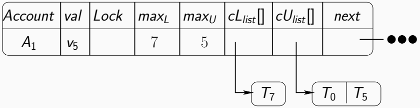


Figure 23: Data structure of BG

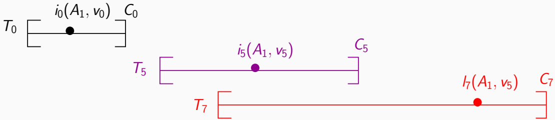
- OSTMs[11] have fewer conflicts than RWSTMs which in turn, allows validators to execute more SCTs concurrently.
- This also reduces the size of the BG leading to a smaller communication cost than RWSTMs.

[11] Herlihy, M., Koskinen, E.: Transactional Boosting: A Methodology for Highly-concurrent Transactional Objects. PPoPP, 2008.

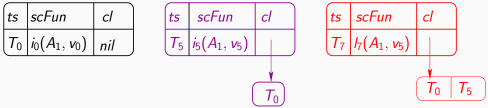
ObjSC: Data Structure of SVOSTM to Maintain Conflicts



(a) Structure of Shared data-item



(b) Timeline View



(c) Transactions Conflict List

Figure 24: Underlying Data Structure of SVOSTM

ObjSC: Single-version v/s Multi-version OSTMs

- *Multi-version OSTMs (MVOSTMs)* maintain multiple versions for each shared data item (object) and provide greater concurrency relative to traditional *single-version OSTMs (SVOSTMs)*.

- *Multi-version OSTMs (MVOSTMs)* maintain multiple versions for each shared data item (object) and provide greater concurrency relative to traditional *single-version OSTMs (SVOSTMs)*.

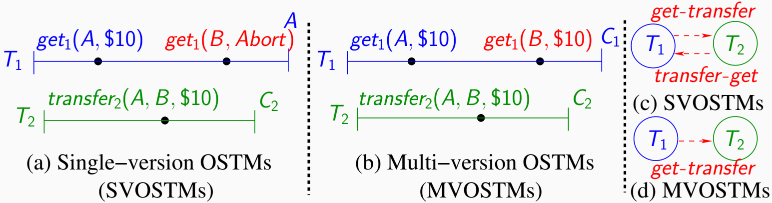


Figure 25: (a) Transaction T_1 gets the balance of two accounts A and B (both initially \$10), while transaction T_2 transfers \$10 from A to B and T_1 aborts. Since, its conflict graph has a cycle (see (c)); (b) When T_1 and T_2 are executed by MVOSTM, T_1 can read the old versions of A and B. This can be serialized, as shown in (d).

ObjSC: Multi-Version OSTM based Miner

- *Multi-Version OSTMs (MVOSTMs)*[12] maintain multiple versions for each shared data item and provide greater concurrency relative to *Single-Version OSTMs (SVOSTMs)*.

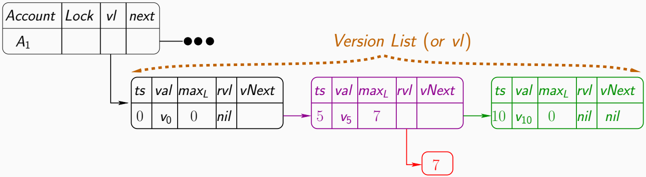
12 Juyal, C., Kulkarni, S., Kumari, S., Peri, S., Somani, A.: An innovative approach to achieve compositionality efficiently using multi-version object based transactional systems. SSS, 2018.

ObjSC: Multi-Version OSTM based Miner

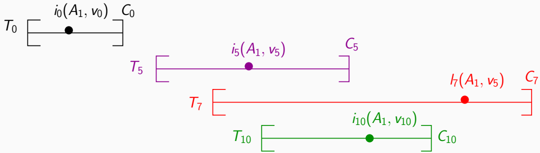
- *Multi-Version OSTMs (MVOSTMs)*[12] maintain multiple versions for each shared data item and provide greater concurrency relative to *Single-Version OSTMs (SVOSTMs)*.
- MVOSTM-based BG has fewer edges than an SVOSTM-based BG, and further reduces the size of the BG leading to a smaller communication cost.

[12] Juyal, C., Kulkarni, S., Kumari, S., Peri, S., Somani, A.: An innovative approach to achieve compositionality efficiently using multi-version object based transactional systems. SSS, 2018.

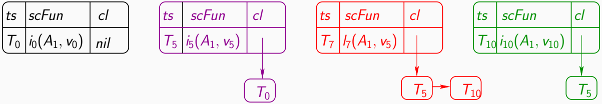
ObjSC: Data Structure of MVOSTM to Maintain Conflicts



(a) Structure of Shared data-item with Version List



(b) Timeline View



(c) Transactions Conflict List

Figure 26: Underlying Data Structure of SVOSTM

ObjSC Correctness Criteria: Opacity

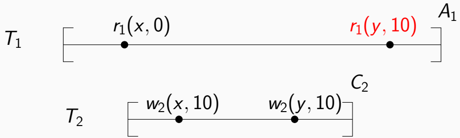


Figure 27: History H is not Opaque

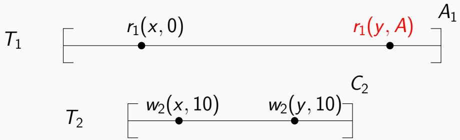


Figure 28: Opaque History H

ObjSC: Working of multi-threaded validator

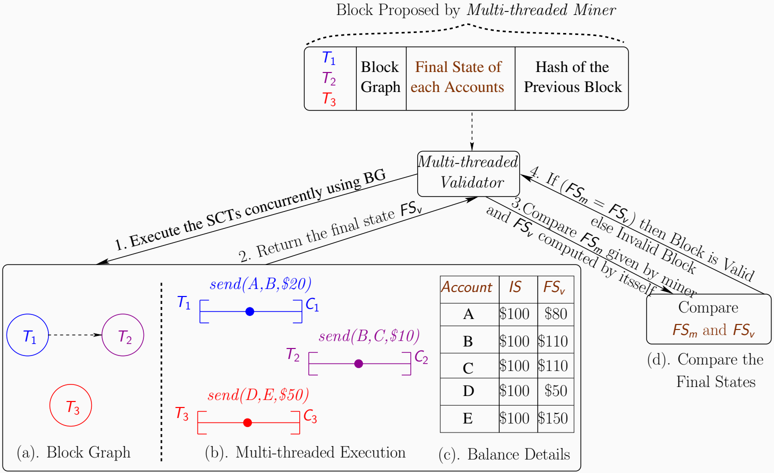


Figure 29: Working of multi-threaded validator

ObjSC: Smart Multi-threaded Validator (SMV)

SMV maintains two global counters (gUC: global update counter and gLC: global lookup counter) and two local counters (lUC and lLC) for each shared data item k to identifies the EMB error.

ObjSC: Smart Multi-threaded Validator (SMV)

SMV maintains two global counters (gUC: global update counter and gLC: global lookup counter) and two local counters (lUC and lLC) for each shared data item k to identifies the EMB error.

Lookup(k):

- **If**(k.gUC == k.lUC)
  1. Atomically increment the global lookup counter, k.gLC.
  2. Increment k.lLC by 1.
  3. Lookup key k from a shared memory.

  else miner is malicious.

ObjSC: Smart Multi-threaded Validator (SMV)

SMV maintains two global counters (gUC: global update counter and gLC: global lookup counter) and two local counters (lUC and lLC) for each shared data item k to identifies the EMB error.

Lookup(k):

- **If**(k.gUC == k.lUC)
  1. Atomically increment the global lookup counter, k.gLC.
  2. Increment k.lLC by 1.
  3. Lookup key k from a shared memory.

  else miner is malicious.

Insert(k, v)/Delete(k):

- **If**(k.gLC == k.lLC && k.gUC == k.lUC)
  1. Atomically increment the global update counter, k.gUC.
  2. Increment k.lUC by 1.
  3. Insert/delete key k to/from shared memory.

  else miner is malicious.

ObjSC: SMV Counter Based Solution

Algorithm 1: SMV(scFun): Execute scFun with atomic global lookup/update counter.

```
// scFun is a list of steps.
while (scFun.steps.hasNext()) do
    curStep = scFun.steps.next(); //Get the next step to execute.
    switch (curStep) do
        case lookup(k): do
            // Check for update counter (uc) value.
            if (k.gUC == k.lUC_i) then
                Atomically increment the global lookup counter, k.gLC;
                Increment k.lLC_i by 1;//Maintain k.lLC_i in transaction local log.
                Lookup k from a shared memory;
            end
            else
                return ⟨Miner is malicious⟩;
            end
        end
        case insert(k, v): do
            // Check lookup/update counter value.
            if ((k.gLC == k.lLC_i) && (k.gUC == k.lUC_i)) then
                Atomically increment the global update counter, k.gUC;
                Increment k.lUC_i by 1;//Maintain k.lUC_i in transaction local log.
                Insert k in shared memory with value v;
            end
            else
                return ⟨Miner is malicious⟩;
            end
        end
    end
end
Atomically decrements the k.gLC and k.gUC corresponding to each shared data-item key k;
```

ObjSC: SMV Counter Based Solution

```
// scFun is a list of steps.
while (scFun.steps.hasNext()) do
    curStep = scFun.steps.next(); //Get the next step to execute.
    switch (curStep) do
        case delete(k): do
            // Check lookup/update counter value.
            if ((k.gLC == k.lLC_i) && (k.gUC == k.lUC_i)) then
                Atomically increment the global update counter, k.gUC;
                Increment k.lUC_i by 1; //Maintain k.lUC_i in transaction local.
                Delete k in shared memory;
            end
            else
                return ⟨Miner is malicious⟩;
            end
        end
    end
end
Atomically decrements the k.gLC and k.gUC corresponding to each shared data-item key k;
```

ObjSC: Experimental Evaluation (1/2)

- In Ethereum blockchain, smart contracts are written in Solidity language, which runs on Ethereum Virtual Machine (EVM).

ObjSC: Experimental Evaluation (1/2)

- In Ethereum blockchain, smart contracts are written in Solidity language, which runs on Ethereum Virtual Machine (EVM).
- EVM does not supports multi-threading.

ObjSC: Experimental Evaluation (1/2)

- In Ethereum blockchain, smart contracts are written in Solidity language, which runs on Ethereum Virtual Machine (EVM).
- EVM does not supports multi-threading.
- We converted smart contracts from Solidity to **C++** language for multi-threaded execution.

ObjSC: Experimental Evaluation (2/2)

- We consider four benchmark contracts Coin, Ballot, Simple Auction, and Mix from Solidity documentation.

ObjSC: Experimental Evaluation (2/2)

- We consider four benchmark contracts Coin, Ballot, Simple Auction, and Mix from Solidity documentation.

  1. **Coin:** A simple cryptocurrency contract.

ObjSC: Experimental Evaluation (2/2)

- We consider four benchmark contracts Coin, Ballot, Simple Auction, and Mix from Solidity documentation.
    1. **Coin:** A simple cryptocurrency contract.
    2. **Ballot:** An electronic voting contract.

ObjSC: Experimental Evaluation (2/2)

- We consider four benchmark contracts Coin, Ballot, Simple Auction, and Mix from Solidity documentation.

    1. **Coin:** A simple cryptocurrency contract.
    2. **Ballot:** An electronic voting contract.
    3. **Simple Auction:** An online auction contract.

ObjSC: Experimental Evaluation (2/2)

- We consider four benchmark contracts Coin, Ballot, Simple Auction, and Mix from Solidity documentation.

  1. **Coin:** A simple cryptocurrency contract.
  2. **Ballot:** An electronic voting contract.
  3. **Simple Auction:** An online auction contract.
  4. **Mix:** Combination of above three contracts in equal proportion.

ObjSC: Experimental Evaluation (2/2)

- We consider four benchmark contracts Coin, Ballot, Simple Auction, and Mix from Solidity documentation.
    1. **Coin:** A simple cryptocurrency contract.
    2. **Ballot:** An electronic voting contract.
    3. **Simple Auction:** An online auction contract.
    4. **Mix:** Combination of above three contracts in equal proportion.
- We ran our experiments on Intel (R) Xeon (R) CPU E5-2690 that supports 56 hardware threads and 32GB RAM.

ObjSC: Experimental Evaluation (2/2)

- We consider four benchmark contracts Coin, Ballot, Simple Auction, and Mix from Solidity documentation.
    1. **Coin:** A simple cryptocurrency contract.
    2. **Ballot:** An electronic voting contract.
    3. **Simple Auction:** An online auction contract.
    4. **Mix:** Combination of above three contracts in equal proportion.
- We ran our experiments on Intel (R) Xeon (R) CPU E5-2690 that supports 56 hardware threads and 32GB RAM.
- We consider two workloads:

ObjSC: Experimental Evaluation (2/2)

- We consider four benchmark contracts Coin, Ballot, Simple Auction, and Mix from Solidity documentation.

  1. **Coin:** A simple cryptocurrency contract.
  2. **Ballot:** An electronic voting contract.
  3. **Simple Auction:** An online auction contract.
  4. **Mix:** Combination of above three contracts in equal proportion.

- We ran our experiments on Intel (R) Xeon (R) CPU E5-2690 that supports 56 hardware threads and 32GB RAM.
- We consider two workloads:

Workload	SCTs	Threads	Shared data items
Workload 1 (W1)	50 - 300	50	500
Workload 2 (W2)	100	10 - 60	500

ObjSC Results: Multi-threaded Miner Speedup

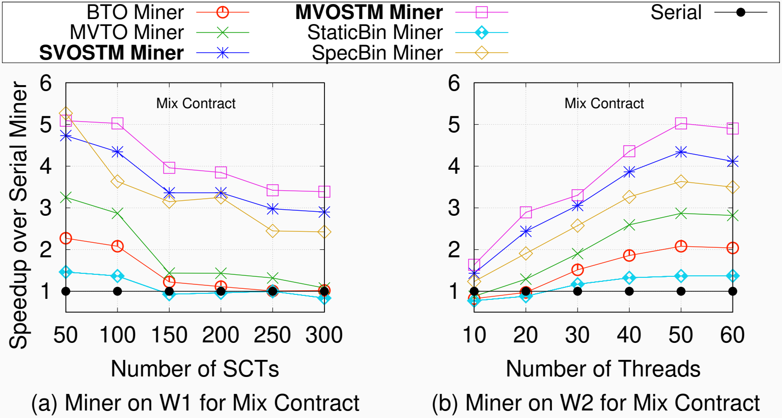


Figure 30: Speedup of Multi-threaded miner over Serial miner

- **MVOSTM, SVOSTM**, MVTO, BTO, Speculative Bin, and Static Bin miner provide an average speedup of **3.91**×**, 3.41**×, 1.98×, 1.5×, 3.02×, and 1.12×, over Serial miner, respectively.

ObjSC Results: Average Speedup by Multi-threaded Miner

Table 2: Overall average speedup on all workloads by multi-threaded miner over serial miner

Contract	Multi-threaded Miner					
	BTO Miner	MVTO Miner	SVOSTM Miner	MVOSTM Miner	StaticBin Miner	SpecBin Miner
Coin	1.596	1.959	4.391	5.572	1.279	6.689
Ballot	0.960	1.065	2.229	2.431	1.175	2.233
Auction	2.305	2.675	3.456	3.881	1.524	2.232
Mix	1.596	2.118	3.425	3.898	1.102	3.080
Total Avg. Speedup	*1.61*	*1.95*	*3.38*	*3.95*	*1.27*	*3.56*

ObjSC Results: SMV Speedup

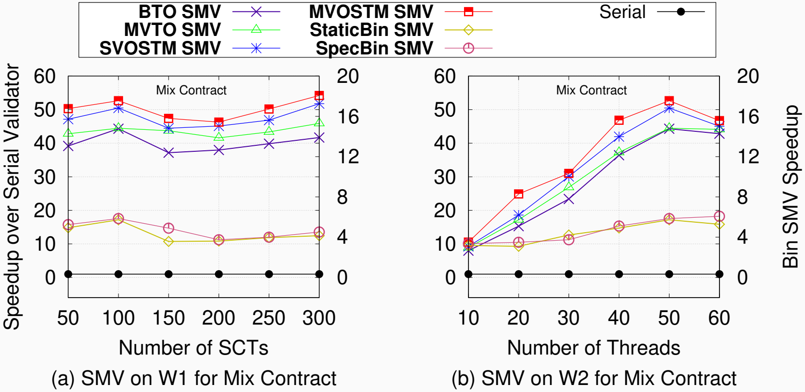


Figure 31: Speedup of SMV over Serial validator

- **MVOSTM, SVOSTM, MVTO, BTO, Speculative Bin,** and **Static Bin Decentralized SMVs** provide an average speedup of **48.45**×, **46.35**×, **43.89**×, **41.44**×, **5.39**×, and **4.81**× over Serial validator, respectively.

ObjSC Results: Average Speedup by SMV

Table 3: Overall average speedup on all workloads by SMV over serial validator

Contract	Smart Multi-threaded Validator (SMV)					
	BTO SMV	**MVTO SMV**	**SVOSTM SMV**	**MVOSTM SMV**	**StaticBin SMV**	**SpecBin SMV**
Coin	26.576	28.635	30.344	32.864	5.296	7.565
Ballot	26.037	28.333	33.695	36.698	3.570	3.780
Auction	27.772	31.781	29.803	32.709	4.694	5.214
Mix	36.279	39.304	42.139	45.332	4.279	4.463
Total Avg. Speedup	*29.17*	*32.01*	*34.00*	*36.90*	*4.46*	*5.26*

ObjSC Results: Malicious Block

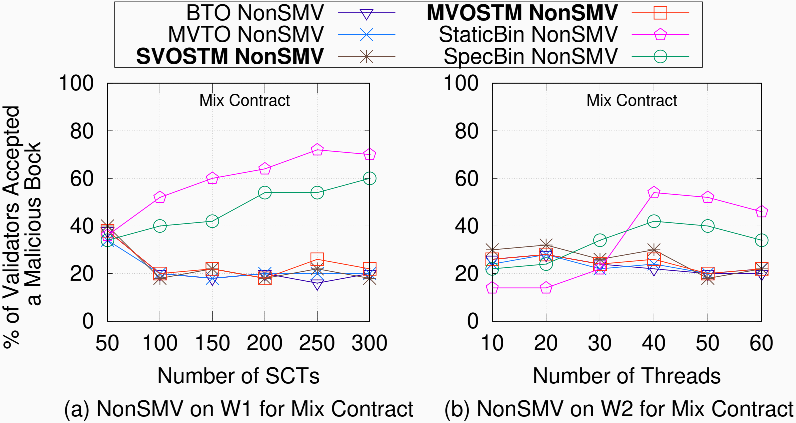


Figure 32: Percentage of NonSMV accepting a malicious block

- Acceptance of even a single malicious block result in the blockchain going into inconsistent state.

ObjSC Results: Dependencies in BG

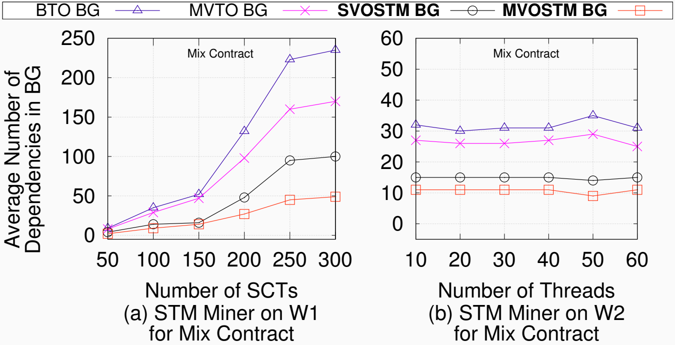


Figure 33: Average number of dependencies in BG for mix contract on W1 and W2

ObjSC Results: BG Depth

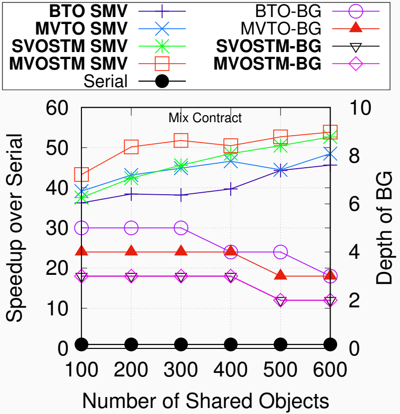


Figure 34: Speedup of SMV over serial and depth of BG for W3

ObjSC Conclusion

- We developed an efficient framework for concurrent execution of SCTs by a multi-threaded miner using two protocols, SVOSTM and MVOSTM of optimistic STMs[13].

[13] Technical report: https://arxiv.org/abs/1904.00358

ObjSC Conclusion

- We developed an efficient framework for concurrent execution of SCTs by a multi-threaded miner using two protocols, SVOSTM and MVOSTM of optimistic STMs[13].
- To avoid FBR errors, the multi-threaded miner captures the dependencies among SCTs in the form of a BG.

[13] Technical report: https://arxiv.org/abs/1904.00358

ObjSC Conclusion

- We developed an efficient framework for concurrent execution of SCTs by a multi-threaded miner using two protocols, SVOSTM and MVOSTM of optimistic STMs[13].
- To avoid FBR errors, the multi-threaded miner captures the dependencies among SCTs in the form of a BG.
- To handle EMB error, we proposed SMV that re-executes SCTs concurrently relying on the BG provided by the miner.

[13] Technical report: https://arxiv.org/abs/1904.00358

ObjSC Conclusion

- We developed an efficient framework for concurrent execution of SCTs by a multi-threaded miner using two protocols, SVOSTM and MVOSTM of optimistic STMs[13].
- To avoid FBR errors, the multi-threaded miner captures the dependencies among SCTs in the form of a BG.
- To handle EMB error, we proposed SMV that re-executes SCTs concurrently relying on the BG provided by the miner.
- The proposed approach achieves significant performance gain over the state-of-the-art SCTs execution framework.

[13] Technical report: https://arxiv.org/abs/1904.00358

ObjSC Future Work

- A malicious miner can intentionally append a BG in a block with additional edges to delay other miners. Preventing such a malicious miner would be an immediate future work.

ObjSC Future Work

- A malicious miner can intentionally append a BG in a block with additional edges to delay other miners. Preventing such a malicious miner would be an immediate future work.
- BG consumes space. So, constructing storage optimal BG is an interesting challenge.

ObjSC Future Work

- A malicious miner can intentionally append a BG in a block with additional edges to delay other miners. Preventing such a malicious miner would be an immediate future work.
- BG consumes space. So, constructing storage optimal BG is an interesting challenge.
- Implementing our proposed approach in other blockchains such as Bitcoin, Hyperledger, and EOSIO is an exciting exercise.

ObjSC Future Work

- A malicious miner can intentionally append a BG in a block with additional edges to delay other miners. Preventing such a malicious miner would be an immediate future work.
- BG consumes space. So, constructing storage optimal BG is an interesting challenge.
- Implementing our proposed approach in other blockchains such as Bitcoin, Hyperledger, and EOSIO is an exciting exercise.
- EVM does not support multi-threading, so, another research direction is to design a multi-threaded EVM.

ObjSC Future Work

- A malicious miner can intentionally append a BG in a block with additional edges to delay other miners. Preventing such a malicious miner would be an immediate future work.
- BG consumes space. So, constructing storage optimal BG is an interesting challenge.
- Implementing our proposed approach in other blockchains such as Bitcoin, Hyperledger, and EOSIO is an exciting exercise.
- EVM does not support multi-threading, so, another research direction is to design a multi-threaded EVM.
- Another interesting direction is to apply concurrency in the nested execution of SCTs.

Collaborators

Sathya Peri
Associate Professor
IIT Hyderabad, India
sathya_p@cse.iith.ac.in

Yogesh Simmhan
Professor
IISc, Bangalore, India
simmhan@iisc.ac.in

Hagit Attiya
Professor
Technion, Israel
hagit@cs.technion.ac.il

Parwat Singh Anjana
Ph.D. Student
IIT Hyderabad, India
cs17resch11004@iith.ac.in

Archit Somani
Postdoc Fellow
Technion, Israel
archit@cs.technion.ac.il

Sweta Kumari
Postdoc Fellow
Technion, Israel
sweta@cs.technion.ac.il

Shrey Baheti
Software Engineer
Cargill Digital Labs, India
shrey_baheti@cargill.com

Sachin Rathor
Software Engineer
Microsoft, India
cs18mtech01002@iith.ac.in

Thanks!