An Efficient Framework for Optimistic Concurrent Execution of Smart Contracts

Presented by:
Sweta Kumari
Ph.D. Student
CSE Department,
Indian Institute of Technology
Hyderabad, India
February 12, 2019

---

The initial idea has been presented in Doctoral Symposium, ICDCN 2019, and awarded with ‘Best Poster Award’.
Team of this Research Work

Parwat Singh
Anjana

Sweta Kumari

Sathya Peri

Sachin Rathor

Archit Somani
Outline of the Talk

Introduction to Blockchain and Smart Contracts

Current Blockchain Design

Bottleneck in Existing Blockchain Design

Challenges to Execute the Smart Contracts Concurrently

Proposed Methodology: Concurrent Miner and Validator

Experimental Evaluation

Conclusion and Future Work
Introduction

- Blockchain
- Smart Contracts
Introduction

Blockchain

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

\[\text{https://bitcoin.org/en/}\]
\[\text{https://www.ethereum.org/}\]
\[\text{https://www.hyperledger.org/}\]
Introduction

Blockchain

• Blockchain is a decentralized, distributed database or ledger of records.
• It maintains information in the form of blocks.

b https://bitcoin.org/en/
c https://www.ethereum.org/
d https://www.hyperledger.org/
Introduction

Blockchain

• Blockchain is a decentralized, distributed database or ledger of records.
• It maintains information in the form of blocks.
• Each block is a collection of transactions and has a pointer to its previous block.

b https://bitcoin.org/en/
c https://www.ethereum.org/
d https://www.hyperledger.org/
Introduction

Blockchain

• Blockchain is a decentralized, distributed database or ledger of records.
• It maintains information in the form of blocks.
• Each block is a collection of transactions and has a pointer to its previous block.
• By design these blocks are immutable but publicly readable.

b https://bitcoin.org/en/
c https://www.ethereum.org/
d https://www.hyperledger.org/
Introduction

Blockchain

- Blockchain is a decentralized, distributed database or ledger of records.
- It maintains information in the form of blocks.
- Each block is a collection of transactions and has a pointer to its previous block.
- By design these blocks are immutable but publicly readable.
- Example: Bitcoin\(^b\), Ethereum\(^c\), and Hyperledger\(^d\) etc.

\(^b\) https://bitcoin.org/en/
\(^c\) https://www.ethereum.org/
\(^d\) https://www.hyperledger.org/
Introduction
Smart contracts

- Modern blockchain systems interpose an additional software layer between clients and the blockchain.
Introduction

Smart contracts

- Modern blockchain systems interpose an additional software layer between clients and the blockchain.
- Client requests are directed to scripts, called *smart contracts*. Examples are Ballot, coin, simple auction etc.
Introduction

Smart contracts

• Modern blockchain systems interpose an additional software layer between clients and the blockchain.

• Client requests are directed to scripts, called *smart contracts*. Examples are Ballot, coin, simple auction etc.

• Smart contracts are similar to a legal contract in which terms are recorded in a legal language.
Introduction

Smart contracts

• Modern blockchain systems interpose an additional software layer between clients and the blockchain.
• Client requests are directed to scripts, called smart contracts. Examples are Ballot, coin, simple auction etc.
• Smart contracts are similar to a legal contract in which terms are recorded in a legal language.
• **Advantages:**
  1. No need for a trusted third party to validate the contract.
  2. Low contracting enforcement.
  3. Compliance costs.
Current Blockchain Design
Ethereum High Level Design

- Ethereum nodes form a peer-to-peer system.
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.
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:* Clients send Transaction T1, T2 and T3 to Miner (Peer4)
Ethereum High Level Design

Figure: Miner forms a block B4 and computes final state (FS) sequentially
Ethereum High Level Design

Figure: Miner broadcasts the block B4
Ethereum High Level Design

Figure: Validators (Peer 1, 2 and 3) compute current state (CS) sequentially
Ethereum High Level Design

Figure: Validators verify the FS and reach the consensus protocol
Ethereum High Level Design

Figure: Block B4 successfully added to the blockchain
Bottleneck in Existing Blockchain
Bottleneck in Existing Blockchain

Ethereum

• In the current era of multi-core processors, serial execution of the transactions by the miners and validators fail to utilize the cores properly and as a result, have poor throughput.
In the current era of multi-core processors, serial execution of the transactions by the miners and validators fail to utilize the cores properly and as a result, have poor throughput.

\[
T_1 \quad w(x, 10) \quad C_1
\]

\[
T_2 \quad w(y, 20) \quad C_2
\]

(a) Serial Execution of transactions

(b) Concurrent Execution

**Figure:** Motivation towards concurrent execution over serial
Bottleneck in Existing Blockchain

Ethereum

- In the current era of multi-core processors, serial execution of the transactions by the miners and validators fail to utilize the cores properly and as a result, have poor throughput.

(a) Serial Execution of transactions

(b) Concurrent Execution

Figure: Motivation towards concurrent execution over serial

- By adding concurrency to smart contracts execution, we can achieve better efficiency and higher throughput.
Challenges to Execute the Smart Contracts Concurrently
Challenges (1/2)

- Concurrent execution of smart contracts transactions may access the conflicting shared data.
Challenges (1/2)

- Concurrent execution of smart contracts transactions may access the conflicting shared data.
- Identifying the conflicts at run-time is not straightforward.
Challenges (1/2)

• Concurrent execution of smart contracts transactions may access the conflicting shared data.

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

• Improper use of locks may lead to deadlock.
Challenges (1/2)

- Concurrent execution of smart contracts transactions may access the conflicting shared data.
- 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 smart contract transactions is difficult.
Challenges (2/2)

- Concurrent execution of smart contracts transactions by validator may incorrectly reject the valid block proposed by the miner.
Challenges (2/2)

- Concurrent execution of smart contracts transactions by validator may incorrectly reject the valid block proposed by the miner.

(a) Concurrent transactions
(b) Equivalent execution by miner
(c) Equivalent execution by validator

Figure: Concurrent execution of transactions by miner and validator
Proposed Methodology: Concurrent Miner and Validator
Proposed Methodology

Concurrent Miner

- We develop an efficient framework to execute the smart contract transactions concurrently by miner using optimistic Software Transactional Memory systems (STMs).
Software Transactional Memory Systems (STMs)

- STMs are a convenient programming interface for a programmer to access shared memory using concurrent threads without worrying about concurrency issues.
Software Transactional Memory Systems (STMs)

• STMs are a convenient programming interface for a programmer to access shared memory using concurrent threads without worrying about concurrency issues.

• Traditionally, STMs export the following methods:
  ○ t_begin()
  ○ t_read()
  ○ t_write()
  ○ tryC() and tryA()
Algorithm 1 Insert($LL, e$): Invoked by a thread to insert an element $e$ into a linked-list $LL$. This method is implemented using transactions.

```
1: retry = 0;
2: while (true) do
3:   id = t_begin(retry);
4:   ...
5:   ...
6:   v = t_read(id, x); // reads the value of $x$ as $v$
7:   ...
8:   ...
9:   t_write(id, x, v'); // writes a value $v'$ to $x$
10:  ...
11:  ...
12:  ret = tryC(id); // tryC can return commit or abort
13:  if (ret == commit) then
14:     break;
15:  else
16:     retry++;
```
IITH-STM Library

- We have used two protocols implemented in IITH-STM library for concurrent execution of the smart contracts by miner.
Basic Time-stamp Ordering (BTO) Protocol

- 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 $ts(t_i) < ts(t_j)$.

![Diagram of BTO](image)

**Figure**: BTO

---

Multi-Version Time-stamp Ordering (MVTO) Protocol

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

![Diagram of MVTO protocol]

*Figure: BTO*

---

\[ f \]

Multi-Version Time-stamp Ordering (MVTO) Protocol

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

Figure: BTO

Figure: MVTO

---

The miner generates a Block Graph (BG) to capture the dependencies between the smart contract transactions.

BG used by validator to execute smart contract transactions concurrently.

Two transactions which do not have a path should be able to execute concurrently.
• The miner generates a Block Graph (BG) to capture the dependencies between the smart contract transactions.

• BG used by validator to execute smart contract transactions concurrently.
The miner generates a Block Graph (BG) to capture the dependencies between the smart contract transactions.

BG used by validator to execute smart contract transactions concurrently.

Two transactions which do not have a path should be able to execute concurrently.
Block Graph Components (2/6)

- The vertices correspond only to committed transactions (or atomic units).
Block Graph Components (2/6)

- The vertices correspond only to committed transactions (or atomic units).
- The edges of the block graph depend on the STM protocol.

For BTO:
- Uses single version and satisfies co-opacity (similar to CSR).
- Hence, conflict graph is same as block graph.
- If $T_i, T_j$ are conflicting and $i < j$ then add an edge from $T_i$ to $T_j$ in the graph.
The vertices correspond only to committed transactions (or atomic units).

The edges of the block graph depends on the STM protocol.

For BTO:

- Uses single version and satisfies co-opacity (similar to CSR).
- Hence, conflict graph is same as block graph.
- If $T_i, T_j$ are conflicting and $i < j$ then add an edge from $T_i$ to $T_j$ in the graph.
• MVTO uses multiple versions and satisfies opacity.
- MVTO uses multiple versions and satisfies opacity.
- Here, block graph is same as multi-version serialization graph.
Block Graph Edges (3/6)

- MVTO uses multiple versions and satisfies opacity.
- Here, block graph is same as multi-version serialization graph.

- Edges in MVTO:
  - *Real Time Edge*: Commit/Termination of a transaction $T_i < \text{beginning of transaction } T_j$. Then Edge goes from $T_i \rightarrow T_j$. 
• MVTO uses multiple versions and satisfies opacity.

• Here, block graph is same as multi-version serialization graph.

• Edges in MVTO:
  
  ○ *Real Time Edge*: Commit/Termination of a transaction $T_i < \text{begining of transaction } T_j$. Then Edge goes from $T_i \rightarrow T_j$.
  
  ○ *Read From Edge*: A transaction $T_j$ reading from transaction $T_i$. Then Edge goes from $T_i \rightarrow T_j$. 

Block Graph Edges (3/6)

• MVTO uses multiple versions and satisfies opacity.

• Here, block graph is same as multi-version serialization graph.

• Edges in MVTO:
  
  ○ *Real Time Edge*: Commit/Termination of a transaction $T_i < \text{beginning}$ of transaction $T_j$. Then Edge goes from $T_i \rightarrow T_j$.

  ○ *Read From Edge*: A transaction $T_j$ reading from transaction $T_i$. Then Edge goes from $T_i \rightarrow T_j$.

  ○ *Version Order Edge*: It captures the multiversion relations among the transactions and form the edges.
Block Graph (4/6)
Data Structure of Lock-free Concurrent Block Graph

(a) Underlying representation of Block Graph

(b) Block Graph

Figure: Pictorial representation of Block Graph
Block Graph (5/6)
Lock-Free Concurrent BG Methods

Concurrent Miner Methods: `addVert()`, `addEdge()`
Concurrent Miner Methods: `addVert()`, `addEdge()`

- `addVert()`: Adds vertices to the graph.
Block Graph (5/6)

Lock-Free Concurrent BG Methods

Concurrent Miner Methods: `addVert()`, `addEdge()`

- `addVert()`: Adds vertices to the graph.
- `addEdge()`: Adds edges to the graph.
Concurrent Validator Methods: searchGlobal(), decInCount()
Concurrent Validator Methods: searchGlobal(), decInCount()

- searchGlobal(): Searches for a vertex with no incoming edges & claims it.
Concurrent Validator Methods: searchGlobal(), decInCount()

• searchGlobal(): Searches for a vertex with no incoming edges & claims it.

• decInCount(): Decrements the in-degree of a vertex.
Experimental Evaluation
Experimental Evaluation (1/3)

- Smart Contracts are written in Solidity language which runs on Ethereum Virtual Machine (EVM).
Experimental Evaluation (1/3)

- Smart Contracts are written in Solidity language which runs on Ethereum Virtual Machine (EVM).
- EVM doesn’t support multi-threading.
Experimental Evaluation (1/3)

- Smart Contracts are written in Solidity language which runs on Ethereum Virtual Machine (EVM).
- EVM doesn’t supports multi-threading.
- We converted smart contracts from solidity to C++ language for concurrent execution.
Experimental Evaluation (2/3)

- We consider four benchmarks Simple Auction, Coin, Ballot, Mixed contracts from Solidity documentation.
Experimental Evaluation (2/3)

• We consider four benchmarks Simple Auction, Coin, Ballot, Mixed contracts from Solidity documentation.

• Simple Auction: Multiple bidders will take the part in bid and at the end of the auction, a bidder with the highest amount will be successful.
Experimental Evaluation (2/3)

- We consider four benchmarks Simple Auction, Coin, Ballot, Mixed contracts from Solidity documentation.

- Simple Auction: Multiple bidders will take the part in bid and at the end of the auction, a bidder with the highest amount will be successful.

- Coin: It is the simplest form of a cryptocurrency. Whoever having account with sufficient balance can send coins from his account to another account.
Experimental Evaluation (2/3)

• We consider four benchmarks Simple Auction, Coin, Ballot, Mixed contracts from Solidity documentation.

• Simple Auction: Multiple bidders will take the part in bid and at the end of the auction, a bidder with the highest amount will be successful.

• Coin: It is the simplest form of a cryptocurrency. Whoever having account with sufficient balance can send coins from his account to another account.

• Ballot: It implements an electronic voting contract in which voters cast their vote to the proposal.
Experimental Evaluation (2/3)

• We consider four benchmarks Simple Auction, Coin, Ballot, Mixed contracts from Solidity documentation.

• Simple Auction: Multiple bidders will take the part in bid and at the end of the auction, a bidder with the highest amount will be successful.

• Coin: It is the simplest form of a cryptocurrency. Whoever having account with sufficient balance can send coins from his account to another account.

• Ballot: It implements an electronic voting contract in which voters cast their vote to the proposal.

• Mixed: Combination of above three contracts in equal proportion.
Experimental Evaluation (3/3)

• We run our experiment for two workloads:
  1. The number of transactions (or atomic-units) varies from 50 to 400, while threads and shared data-objects are fixed to 50 and 40 respectively.
  2. The number of threads varies from 10 to 60 while atomic-units are fixed to 400 and shared data-objects to 40.
Experimental Evaluation (3/3)

- We run our experiment for two workloads:
  1. The number of transactions (or atomic-units) varies from 50 to 400, while threads and shared data-objects are fixed to 50 and 40 respectively.
  2. The number of threads varies from 10 to 60 while atomic-units are fixed to 400 and shared data-objects to 40.

- Concurrent Validator used two approaches: 1) Fork-Join Approach and 2) Decentralized Approach.
Simple Auction Contracts

Figure: Simple Auction Contracts

- It achieve 3.9x and 4.45x average speedups for concurrent miner using BTO and MVTO STM protocol respectively. Along with BTO and MVTO validator outperform average 35.8x and 43.1x than serial validator respectively.
Coin Contracts

- It achieves 4.7x and 5.2x average speedups for concurrent miner using BTO and MVTO STM protocol respectively. Along with BTO and MVTO validator outperform average 25.6x and 31.3x than serial validator respectively.

Figure: Coin Contracts
Ballot Contracts

- It achieve 3.1x and 2.8x average speedups for concurrent miner using BTO and MVTO STM protocol respectively. Along with BTO and MVTO validator outperform average 55.9x and 62.9x than serial validator respectively.
Mixed Contracts

Figure: Mixed Contract

- It achieve 3.0x and 3.7x average speedups for concurrent miner using BTO and MVTO STM protocol respectively. Along with BTO and MVTO validator outperform average 52.3x and 59.1x than serial validator respectively.
Conclusion and Future Work
Conclusion

• Introduced a novel way to execute the smart contract concurrently by miner using optimistic STMs.

---

Conclusion

- Introduced a novel way to execute the smart contract concurrently by miner using optimistic STMs.
- Implemented the concurrent miner with the help of BTO and MVTO but its generic to any STM protocol.

Conclusion

• Introduced a novel way to execute the smart contract concurrently by miner using optimistic STMs.

• Implemented the concurrent miner with the help of BTO and MVTO but its generic to any STM protocol.

• We proposed a lock-free graph library to generate the block graph.

Conclusion

- Introduced a novel way to execute the smart contract concurrently by miner using optimistic STMs.
- Implemented the concurrent miner with the help of BTO and MVTO but its generic to any STM protocol.
- We proposed a lock-free graph library to generate the block graph.
- We proposed concurrent validator that re-executes the smart contract transactions deterministically and efficiently with the help of block graph given by concurrent miner.

\[g\] Technical report: https://arxiv.org/abs/1809.01326
Conclusion

- Introduced a novel way to execute the smart contract concurrently by miner using optimistic STMs.

- Implemented the concurrent miner with the help of BTO and MVTO but its generic to any STM protocol.

- We proposed a lock-free graph library to generate the block graph.

- We proposed concurrent validator that re-executes the smart contract transactions deterministically and efficiently with the help of block graph given by concurrent miner.

- Proposed protocol shows significant performance gain as compared to serial miner and validator.

---

Future Work

- To achieve the greater concurrency, we are working on *Object-Based STM* instead of *Read-Write STM*.
Future Work

- To achieve the greater concurrency, we are working on Object-Based STM instead of Read-Write STM.

- This idea can be applied to other blockchains such as Hyperledger, Ripple, Quorum etc as well.
Thanks & Questions Please!!