Execute smart contract transactions concurrently using optimistic STMs.

Om Sitapara
Dept. of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
cs16btech11036@iith.ac.in

Harshit Patel
Dept. of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
cs16btech11017@iith.ac.in

Dr.Sathya Peri
Dept. of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
sathya_p@iith.ac.in

Parwat Singh Anjana
Dept. of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
cs17resch11004@iith.ac.in

Abstract*—Generally, a block of the block-chain consists of multiple transactions of smart contracts which are added by a miner. These transactions of smart contracts are executed sequentially by miners to append a correct block into the blockchain. In most of the current day blockchains the miners and validators execute the smart contract transactions serially. After the miners have mined a block it is validated by the validators. The validators re-execute the transactions using smart contract, serially. If the final state of the block formed by the miners is same as the final state recorded by the validators, the block is validated and is added to the blockchain. This addition to blockchain follows a consensus protocol. Currently in all the existing blockchains the smart contract transactions are executed serially both by miners and validators. Due to this approach we are not able to utilize the benefits of multi-core processors, thereby resulting in poor throughput. Our objective is to add concurrency to these smart chain executions, utilizing multiple cores thereby achieving higher throughput and increasing efficiency. *The paper objective is to execute smart contract transactions concurrently using optimistic Software Transactional Memory systems(STMs). To test performance and efficiency, by integrating STM with an existing blockchain.

I. Introduction

Software Transaction Memory Systems(STMs) [1] are a convenient programming interface for a programmer to access shared memory using concurrent threads without worrying about concurrency issues. Currently in most commonly used blockchains the smart contract transactions are executed serially (both miner and validator) resulting in poor throughput.

STM will be used to take care of synchronization issues among the transactions and ensure atomicity (serializability). Using multi-threading the validators also execute the transactions, then it may be the case that the validators arrive at a final state which is different from the final state arrived by the miners. This concurrency issue is solved using STM. STM will generate a block graph and store it into the block. The block graph is the conflict representation of the transactions which are executed by a block producer in a proposed block. The block graph is generated concurrently as the transactions are executed by different threads. During the validation phase, this block graph will be used by validators to determine a order to execute the transactions utilizing multiple cores. The block graph will ensure the same final state reached by the producers and validators.If the validation is successful then proposed block is appended into the blockchain and miner gets incentive depending on the blockchain architecture otherwise the proposed block is discarded.

TABLE I
Comparison between common blockchains

Blockchain	Number of transactions per second
Bitcoin	7
Ethereum	15
IOTA (Tangle)	50
EOSIO	3000
Visa	24000

This table demonstrates the number of transactions executed per second by different blockchains. As seen from the figure, these are very low. This is one of the biggest problem hindering the scalability of blockchains. Our approach in this paper is to study different blockchains

and try to intergrate optimisitic STM with the transaction processing phase, thereby providing a scalable solution. Integrating STM with any blockchain can lead to additional overhead due to invoking of STM functions, retrieving dependency between transactions, creating and storing block graph, passing blockgraph to validators, validators validating according to the block graph etc.

II. Objective

To execute transactions concurrently using STM and to study the trade-off between the speed-up obtained on concurrent execution of transactions and the overhead of incorporating STM with the blockchain flow.

III. Blockchains Studied

A. IOTA (Tangle)

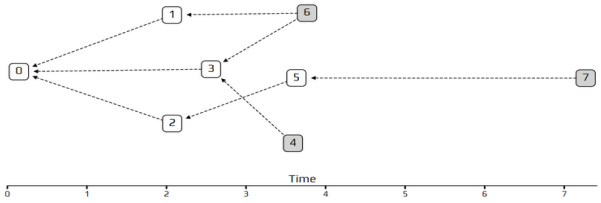


Fig. 1. Example of a BlockDAG. [6]

1) Architecture [2]: Tangle is the data structure behind IOTA. In Tangle there is no concept of blocks or blockchain. The Tangle is a DAG, where vertices represent transactions, and edges represent approvals. When a new transaction is introduced, it is added as a new vertex in the DAG. The new vertex is attached to other two old vertex which is said to approve the old transaction. The process of selecting this two vertex is by performing a random walk with bias. The random walk is done twice to select two vertex which the current incoming transaction is processing and validating In IOTA there are no miners or validators. To add a transaction in the tangle, it has to validate two previous transactions which are represented by direct edges. The main idea is that for a user to issue a transaction the user must work to approve other transactions. In IOTA each transaction is a vertex in the DAG. Each of this transaction has a conformation confidence which shows that how likely is that transaction confirmed in the DAG. This conformation confidence is used to mitigate problem like double-spend. [7]

2) Advantages: The tangle does not have a built in maximum throughput. So its highly scalable. As there are no miners there is no concept of miner fees, thus each transaction sends the full input amount.

3) Disadvantages: The main reason is that each vertex in the DAG represents only one transactions, so STM cannot be used to parallelize the transactions as we cannot store the serialization graph in the nodes of the Tangle. The smart contracts are written in a new language called Abra also which is runs on new platform called Qubic so integrating parallel and concurrent features in new packages is very difficult. As currently there are not many Dapps running on IOTA, the number of transactions coming in the network are very less a special coordinator algorithm is required to maintain the authenticity of the transactions coming.

B. EOSIO

1) Architecture [5]): EOSIO provides a decentralized platform with characteristics like an operating system, that uses blockchain to maintain a distributed, trustless ledger of events and transactions happening on this platform. To decide who creates new blocks (Consensus) is achieved using delegated proof-of-stake, which involves staking tokens to get the right for voting for block Producers, which are full nodes that adds new blocks to the blockchain. Voters can remove block producers and vote for new ones if they suspect any malicious activity. At a given time, only 21 block producers create new blocks. The transaction speed becomes very fast due to less number of verification like proof-of-work based consensus algorithms. Remaining delegates who get fewer votes than 21 block producers can get chance to produce blocks when any of the 21 block producers are voted out by community at any time. These delegates are known as standby producers. Each block is produced exactly in 0.5 seconds. If producers don't produce blocks at that 0.5 seconds, then no other producer shouldn't produce at that time slot. For producing a block the block producer takes the transaction from a pending incoming transaction queue and process it using process transaction sequentially. It is invalid for a block producer to produce a block at any other time slot than one they are scheduled for.If a block producer misses to produce the block in the given time, it is removed from the producers list. EOSIO enables creation of smart contracts, whose execution and resource consumption is handled just like a typical application running on an OS. The Eosio smart contract compiler eosio-cpp which is an extended c++ compiler used to convert this c++ code into web assembly(WASM) which will be further integrated with producer code. Eosio uses tables called multi-index tables as the main way to store data and has actions to interact with them. Since the requirement for CPU and bandwidth is transient, as it will be required only till the

execution of some smart contract action, these resources are obtained by staking tokens for some amount of time. RAM is a persistent storage should be purchased beforehand according to the need and this persistent information is not stored on the blockchain. Blockchain is only used to record transactions and also the events that indicate changes in the persistent information of smart contracts.

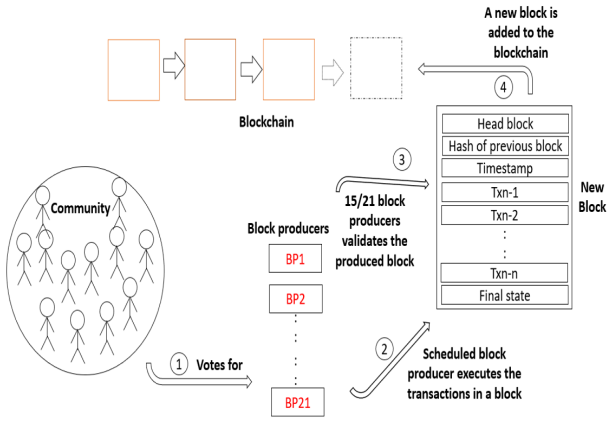


Fig. 2. Pictorial representation of production of a block in EOSIO.

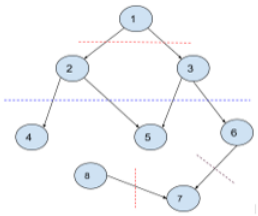
2) Consensus [4]: Consensus Protocol: DPOS-3.0 In DPOS 3.0 last irreversible block is defined which is the most recent block that 2/3+1 of block producers have built off of. If 2/3+1 of producers have built on a chain that confirmed a block then it is likely impossible for there to be any other fork.

3) EOSIO development Ecosystem [3]:

- **Nodeos** is the core EOSIO node daemon that can be configured with plugins to run a node. It will act as a local node which can be used for development and testing purposes, although it can be configured to use as a full node and even for block production.
- **Cleos** is the command line interface to interact with the local node daemon, and can be configured to interact with a remote node too. It is used to issue commands relating to managing wallet, configuring node and issuing transactions to smart contracts.
- **Keosd** is the component that securely stores EOSIO keys in wallets.

IV. Methodology

STM produces a conflict-list ensuring serializability using which we create a Block graph that captures the conflict relations among the transactions. Block graph is generated concurrently as the transactions are executed. This block graph will further be used by the validators to execute the transactions and reach the same final state as the miners. Miner stores this block graph in the block and pass it to the validators. At the validator side this graph is parsed as : all the source node (each node represents a transaction) are executed concurrently. Once a node from a graph is executed the edges from that node removed resulting in new source node. This is performed until all the nodes in the graph are executed. A sample execution of the transactions via Block graph is shown in Fig 3.



Phase No	Transactions executed Concurrently
1	1,8
2	2,3
3	4,5,6
4	7

Fig. 3. Example of execution of transactions using Block-Graph.

V. Experimental Details

IOTA(tangle) uses cubic platform with Abra language and EOSIO uses C++ for smart contracts. We prefer EOSIO as the our STM implementation was also in C++. EOSIO unlike tangle contained multiple transactions in a block which was inherently needed for us. Setup the new experimental EOSIO blockchain network and Esoio.cdt (contract development toolkit). Modify the EOSIO blockchain such that it will add the block graph to the block proposed. Integrating the lock-free graph library with EOSIO. Integrating STM library with Eosio. Writing an on chain smart contract and our process transaction function which will process the transaction of this smart contract. We created a smart contract to print “Hello username” and deployed it. Then we measured the time taken by each transaction of this smart contract to execute. Now we developed a smart contact on chain and made the function to do the same task thereby simulating the exact working of off-chain smart contract. Implemented a new method to execute this on-chain smart contracts transactions both sequentially and concurrently.

1) Challenges: Integration of the STM library with Eosio-cdt : We were not able to integrate the STM library in the Eosio-cdt because the smart contracts in eosio are compiled by eosio-cpp an extended version of c++ compilers which converts the smart contract code to wasm (web assembly code). Now STM internally uses atomic instructions of c++ to assure correctness but there are no atomic instructions available in wasm so we were not able to integrate the stm library in the Eosio.cdt.

VI. Experimental Evaluation

- Fig 4 shows the result obtained by stimulating STM. [1]
- From Fig 6 we see that:
  - Time taken by for one transaction using off chain smart contract : 3.3ms-3.4ms
  - Number of transactions being processed in one block sequentially : 145-150
  - Number of transactions being processed in one block concurrently: 1100-1200

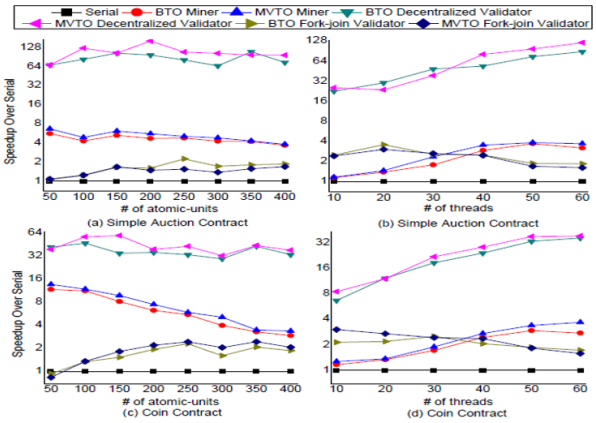


Fig. 4. Results obtained by simulation of STM.

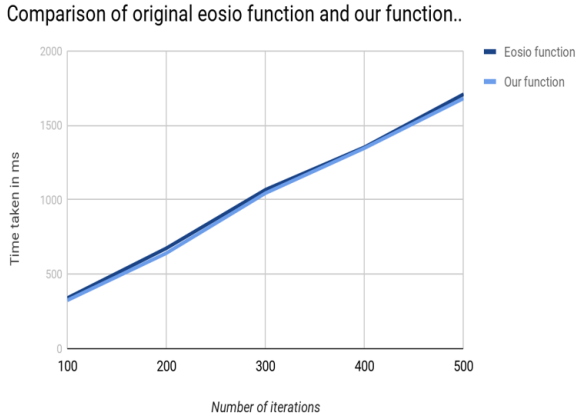


Fig. 5. Results obtained by simulation of STM.

VII. Conclusions

- The result of STM simulation shows that it can achieve higher throughput by providing an efficient framework to parallelize smart contract transactions.
- Thus we conclude that the results obtained by parallelizing the new implemented function will be same as parallelizing the already written function is EOSIO.
- IOTA contains a single transaction in a block, hence cannot be integrated with STM.

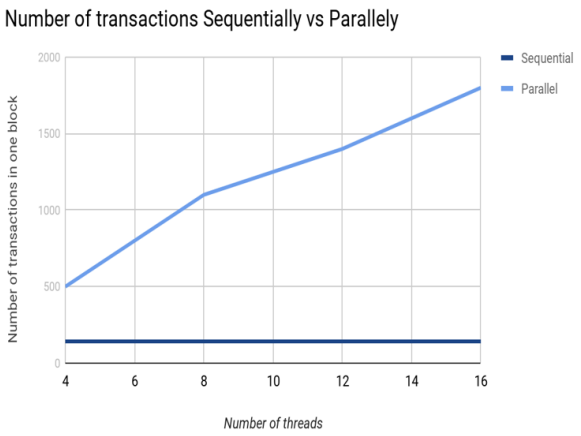


Fig. 6. Results obtained by running new function parallel.

VIII. Future Work

- To write a real world smart contract with STM and try to parallelize it when there are dependency among the transactions.
- Validating the transactions using the block graph.
- Compare the performance and measure the speed up of using the STM inside smart contracts.
- This approach can be extended to any blockchain containing multiple transactions in a block.

References

[1] Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. "An efficient framework for optimistic concurrent execution of smart contracts." In 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pages 83–92. IEEE, 2019

[2] Serguie Popov, "The Tangle", April 30, 2018, Version 1.4.3.

[3] EOSIO Architecture Terminology

[4] DPOS-3.0

[5] EOS.IO Technical White Paper, March 16, 2018 v2

[6] IOTA (Tangle) Image

[7] IOTA-consensus