Parallel Execution of Smart Contracts on Blockchains Without Compromise of Security and Decentralization

Mustafa Aljadery mustafa@mustafaaljadery.com
Independent Researcher

Abstract

Currently, most blockchains run on web-assembly-based run time and can only handle one change to the state at a time. Although novel ideas have been presented concerning Byzantine Fault Tolerance consensus in the past couple of years, a new limit to transaction output has presented itself in the form of execution limitations. The EVM (Ethereum Virtual Machine), the most commonly used execution mechanism for blockchains currently, can only modify the blockchain one contract at a time. A possible solution to the bottleneck that is presented with this execution environment is the execution of contracts in parallel.
In this report, I am going to discuss in detail the most common execution mechanisms of blockchains, currently, and describe possible solutions to the limitation of transaction execution. Instead of the single-threaded web-assembly-based EVM, this report is going to highlight multi-threaded solutions.

1 Introduction

A smart contract can be viewed as a digital contract between two parties that in the event of a conflict, a result is reached by the contents of the smart contracts. Smart contracts have no central authority and live in the blockchain. Typically they are used so that all participants can be immediately certain of an outcome. They can be thought of as simple conditional statements that are immutable.
Proposed changes to a fundamental feature in a blockchain fall under the back-testing of three fundamental principles of a decentralized blockchain. A decentralized blockchain is secure, scalable, and decentralized. Security was the major challenge for the first implementations of a distributed ledger and the main breakthrough was the use of a consensus mechanism. Security is the integrity of the ledger. A ledger must be able to withstand attacks from malicious actors without compromise. Decentralization gives all users of the blockchains the ability to access and modify data in the blockchain without censorship resistance. Finally, scalability refers to the feature that a ledger should optimize for the throughput of transactions. In this report as we propose a solution for scaling the execution of smart contracts, we keep in mind that a solution must also not compromise the decentralization or security of the blockchain.
Although the numerous proposed scaling arguments in the upgraded version of Ethereum (ETH 2.0), ETH 2.0 does not optimize the EVM but rather optimizes the implementation. In ETH 2.0, it is proposed that the EVM will move to eWASM. Web-assembly removes Ethereum's dependence on recompiled smart contracts, thus giving developers the flexibility to develop code in multiple programming languages that execute with better speed. However, the improvement of speed is not a function of better optimization of the execution of the smart contracts, but rather a function of the change from native javascript to the compiled WASM code. Transactions are still processed sequentially, slowing down the system.
A sequential execution model for smart contracts is the main bottleneck for smart contracts on most blockchains. A sequential model may be viewed as the only model since it ensures the blockchain state is not modified maliciously by multiple parties concurrently. However, with the proper security and optimization of multiple cores on a validator, parallel execution can increase the throughput of contract optimization by a large factor.
This paper is only going to discuss briefly the impli-

cation of such a solution to the execution layer on layer 2 scaling. Layer 2 scaling is scaling above the original protocol of the blockchain where transactions are processed on additional chains and the results are posted on the original chain. Although concurrency techniques can significantly increase the throughput of multiple execution mechanisms on layer 2 scaling solutions, this paper is going to focus directly on the execution layer of the problem (layer 1).

2 Background

On the highest level, a blockchain is just a distributed database. All nodes of the database are sharded among different participants and those participants could modify and access the information. The key difference between a blockchain and another shared database is how the information is structured. In a blockchain, the primary data structure is a block and a block can only hold a certain amount of information. Once this limit has been reached, the block is closed and connected to the previous block. Inherently, a blockchain by itself is not secure enough to hold financial transactions. A consensus mechanism, in addition to other secure algorithms, is what provides the blockchain with the necessary security against attacks.

Ethereum is a decentralized distributed ledger whose goal is to give its users more control of their data. The main addition of Ethereum, compared to the original distribute ledger Bitcoin, is the concept of smart contracts. A smart contract is a program that governs the behavior of accounts within the Ethereum blockchain. Currently, the primary way to interact with the blockchain is through the object-oriented language solidity. The smart contract itself is a self-executing an agreement that is reached between the parties through the lines of code that are written in the contract. Smart contracts are also immutable, they cannot be changed once they have been deployed on-chain. The function of a protocol is to give the users access to that data. The protocol must verify that the smart contract is present. Protocols also provide the feature that smart contracts are permissionless. Anyone can write solidity code and deploy it as a smart contract on the Ethereum blockchain.

When optimizing for scaling, many inherent parts of the distributed ledger may be optimized. A more efficient consensus mechanism may achieve the speed of over 100x the current Nakamoto Consensus used in the Ethereum protocol. The consensus I argue is the most important part of any of the protocols. Without consensus, your security and decentralization are compromised and with an inefficient consensus mechanism, the protocol's scaling is inefficient. In this paper I am mostly going to focus on scaling in terms of the execution of smart contracts, however, the understanding consensus is fundamental to understanding optimization in the core security aspect of any protocol.

The environment in which smart contracts are ran is subject to the best optimizations, as that is the home of execution. The Ethereum virtual machine (EVM) only operates on 256 bit integers. The main limit is there to limit the gas used of a transaction on the blockchain. Gas which is calculated in the native currency of ether is the fees a user of the blockchain must pay to interact with the blockchain. Optimizations in the JIT-EVM not only reduce execution time but lower gas for users of the blockchain. On an outer level, this attracts more users to the blockchain.

The fundamental concept of concurrency is present in the execution of smart contracts on the Ethereum blockchain, however, it is by no means optimized. Any execution of a smart contract must be done by all participating nodes in the protocol. This execution is done in parallel, as a synchronous execution of a contract on all of the participating nodes would intensely slow down the system and not achieve any benefit. The additional optimizer in this model is the parallel execution of smart contracts within each node. Currently, each node, although being multicore, can only handle one execution of a smart contract at a time. The proposed methods in this paper indicate that if each node executes smart contracts in parallel, a greater output in terms of execution speed can be reached than a synchronous execution approach.

3 Brief History of Consensus

Consensus is how voters in the blockchain agree on a decision. This ensures that the network state is synchronized. Consensus is what keeps the network safe from malicious attackers. A malicious attacker must take control of the majority of the network for most networks in order to manipulate funds in the distributed ledger.

Consensus is part of the blockchain trilemma. It provides security for the blockchain. No one can steal the funds of the participants in the blockchain. Moreover, many researchers in the past couple of years have looked at optimizing consensus for scalability. The simple idea is that if consensus is reached quicker, the blockchain would be able to processes more data. Before the bitcoin white paper, classical consensus,

more commonly known as Practical Byzantine Fault Tolerance, was based on an all-to-all voting mechanism. A leader must initiate a process for the voting and other participants continue the voting until consensus is reached. Such a consensus protocol is inefficient which regards the desired scalability of a blockchain. Each node has an n^2 overhead.

The Nakamoto Consensus was the next novel improvement to the landscape of consensus. Byzantine Fault Tolerant systems all relied on a leader and if the leader acted maliciously, it would be difficult for the other members of the system to reverse the malicious actions. The Nakamoto consensus algorithm removed the leader in the Byzantine Generals Problem. Consensus is reached without a leader. This is the proof-of-work mechanism. For the users to have a trustless, decentralized network, nodes must contribute their computational power.

The newest consensus algorithm that rivals the Nakamoto Consensus algorithm was published by an anonymous pseudonym called Team Rocket. It proposes the ideas of leaderless Byzantine Fault Tolerance through metastability. In this type of consensus, random sub-sampling is used to reach consensus. The advantages of such a consensus algorithm for a blockchain are low latency, high throughput, and resilience to a majority attack. In the Nakamoto consensus proof-of-work and the proof-of-stake consensus, the voting power of 51 percent of the network would be sufficient to gain the advantage of the network. However, in the metastability, random sub-sampling approach, an attacker could only compromise the system if and only if the attacker owns 80 percent of the network. Through this consensus, irreversible finality can be reached in sub-two seconds. Many times faster than traditional proof-of-work consensus, without compromising the security of the blockchain.

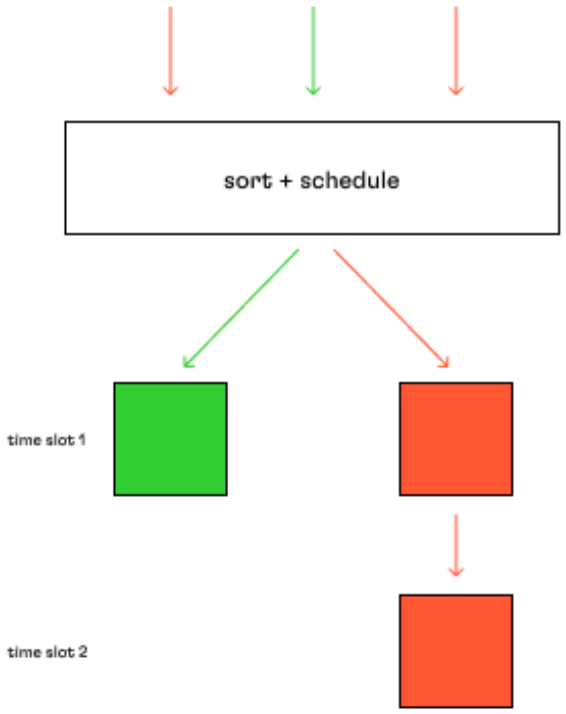
Scaling in the consensus layer improves the transaction per output in terms of block finality rate. Such improvements are fundamental to scaling the blockchain. Before optimizing for execution in smart contracts, optimizing for the best consensus method provides larger scaling for the blockchain.

4 Scheduler Implementation

Many of the current blockchains track account through a local key-value-based storage. A problem with such a mechanism for storing the data of many accounts in a blockchain is that it does not optimize for speed. A read and write to the state has to be processed sequentially. If the state is organized such that the read and the write can be done concurrently, scaling of the system can reach many times its current throughput.

A key solution to this problem is scheduler implementations. Scheduler implementations support any number of tasks, up to which the system can handle, and each task has an attached priority. Furthermore, if the call to the blockchain specifies exactly whether the actions of the call are a read or write function, the virtual machine can prepare the necessary resources concurrently. Such an implementation would significantly speed up execution as instead of the virtual machine processing transactions one after the other, non-conflicting transactions can be bundled. Through this implementation not only will each contract be processed in parallel between the validators, but each contract will be processed in parallel in every single validator.

An implementation of a scheduler will have the virtual machine follow two rules: 1) Each of the incoming transactions is sorted for instances of dependent transactions. 2) Each non-dependent transaction is scheduled and executed in parallel.



4.1 Validators

The main optimization that can be done here is the validators themselves. Through the use of CPU's and

GPU's, the inputs can be sorted so that they can be concurrently executed on-chain. If the instructions that are called to the sorter before the execution on a chain all call the same usage, then all of the transactions can be executed on different cores of the validators. If the block size and the requirements of validators are not important, a GPU would be able to sort through the data as fast as possible. However, adding requirements to validators decreases the level of decentralization when looking at the blockchain trilemma. GPUs are not necessary for concurrency but they can play a big role in quicker sorted data.

4.2 Limitations

This approach is very useful, however, you can not process overlapping transactions in parallel because one modifies the state and then consequent transaction(s) rely on the modified state. Nevertheless, transactions that do not rely on the state of the other are perfect examples of a use case for such an implementation.

4.3 The Perfect Implementation

The perfect implementation of such a system is to distribute overlapping transactions to different shards. This system only works in a sharded blockchain. ETH 2.0 is a perfect example of a blockchain in which this system would produce computation many times the current Ethereum output. Sharding splits a database (in our use case a blockchain) horizontally into different shards that all work to handle the load of the database. Such horizontal scaling increases the transactions per second of the blockchain as the network is democratically split into different shards. ETH 2.0 proposed a blockchain with 64 different shards all working in parallel to handle the congestion of the network. If instructions are organized so that transactions may execute without overlapping in different shards of the blockchain, there would not be a conflict of state and output would be equivalent to the increase of the shards.

4.4 Miscellaneous

ETH 2.0 is not a forward scaling blockchain. A forward scaling blockchain is one where an increase in the number of validators would increase the scalability of the blockchain without the compromise of the other two parts of the blockchain trilemma. Such parallel scaling would make Ethereum dynamically scalable. Furthermore, a common understanding that the validators themselves must have more cores. Thinking about the blockchain trilemma, this would lower the decentralization because then you would require validators with more cores. This would yield the biggest scalability but compromises decentralization. If we only refer to the validator specifications released by the Ethereum organization for an ETH 2.0 node, a processor must be quad-core and the current model can only use a single core at a time. An implementation of a scheduler would take advantage of the other three cores in the validator.

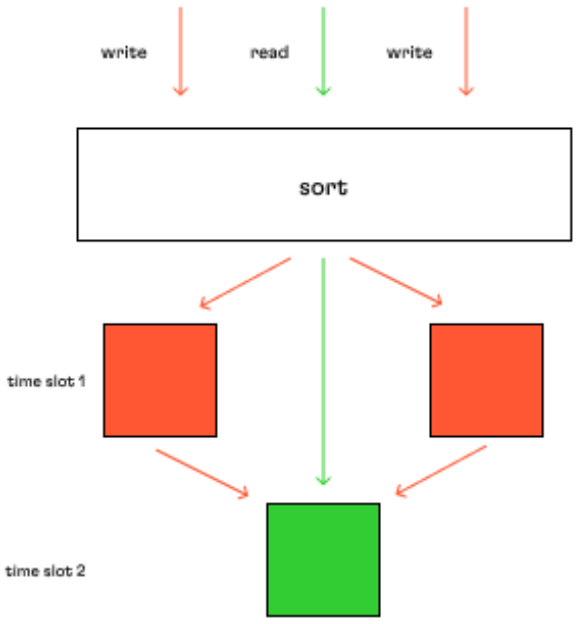
5 Lock-Based Concurrent Data Structure

A lock-based concurrent data structure is a data structure that can be modified by several threads in parallel. This idea is first presented and most commonly used in the Linux Kernel. The sloppy counter in the Linux kernel is used as a way for speeding up increment and decrement operations that are used by a shared variable. A sloppy counter instead of incrementing the global counter increments a local counter. Periodically, the local counter transfers the value of the local counter to the global counter. This is very useful in concurrent, multi-threaded systems as each thread can be implementing another counter. Multiple counters run in parallel and update the global state periodically. The sloppiness of the counter is how often the local-to-global transfer is occurring. In terms of optimization, the bigger the sloppiness, the more scalable the counter. The less the sloppiness, the more the counter acts as a synchronous system. Two big disadvantages of a scalable distributed system that utilizes sloppy counters are the speed of the real value and the accuracy of the global value at any given time. At any given time, especially if the sloppiness of the counter is large, you are subject to the global value not being up to date by the increments of the local values as they have not synchronized yet. This is a sacrifice in the accuracy of the data. Adding locks to the data structure will keep it thread-safe. Locks are present for each local counter and one for the global counter.
On another note about the speed of execution of smart contracts, smart contracts for inherit applications can experience slow down not from the virtual machine execution of the smart contract but poorly written smart contract code. Unoptimized smart contract code may cause delays greater than what can be optimized in a parallel system as dependencies on the state may be increased in which a parallel system would not be applicable as most of the calls will be

dependent on the state.

5.1 Parallel Executing Counters in Validators

When defining the behavior of a smart contract, there are two types of executing functions: read and write. If two write states are in the same block, then they cannot be run in parallel as they are dependent. You can't have two transactions in the block that both modify the state tree. The scheduled implementer was a solution to this problem as those that write the state and are in the same block are not executed at the same time they are synchronized, but the transactions that don't write the same state in the block can be executed asynchronously. An even better idea is to use different storage mechanisms. If you split the network into multiple local counters and large global counters, you can execute conflicting functions in the local counters and pass them into the global counter using a sloppiness function. The different storage entities would not have any conflict executing multiple states writes as they are independent.
This idea is very similar to sharding. ETH 2.0 proposes to split up the blockchain into 64 different shards and the shards are fundamental complete protocols. They implement their state and then write back to the global beacon chain. If the storage for the smart contracts was split into 4 different sub-storages and then each with their executing state, conflicting functions in terms of state modifiers, will be sent to different storage.
To view the actual state and execute a read on the blockchain, all values of the sub-storages (sub-counters in the example that I used) will have to be accessed. This makes the sloppiness dynamic. The sloppy counters that I defined above have a state timeout to when they execute back to the global counter, however, when implemented in the aspect of smart contracts, two functions are possible, a read and a write function. We can partition the write functions, but in terms of the read function, blockchains must get the accurate state or the network may be compromised. All local storage must send back information to the global storage to get accurate information. The speed of such an adjustment to the blockchain relies on the number of reading inputs. The more read inputs the lower the sloppiness in the sloppy counterexample. The slower the run time as it becomes more of synchronous execution.



5.2 Limitation to the Approach

As highlighted in the previous subsection, the number one issue with a lock-based storage approach is the sloppiness of the system. The lower the slopiness of the system the more synchronized the execution of the contracts is. Read execution overhead may cause more synchronization, but it is always at least as good as the synchronized solution or better. In an environment was minimal read executions, the solution is many times faster than the current state of the art. In a situation with many dependent read executions, it is never slower than the synchronized approach. This is also true with the scheduled execution approach. It is never worse than a synchronized approach but has the upside potential of speeds multiple times the current approach.

6 Relevance

An increase in the output of smart contracts not only attracts more users to a decentralized blockchain as the costs of executing a transaction are lower, but it also speeds up the system for more use cases. The largest payment processor visa for example can process 65 thousand transactions per second, while the cap for the current Ethereum implementation is 15

transactions per second. A faster Ethereum would yield more external applications for blockchain protocols. Furthermore, it a more scalable. Scalablity of the blockchain itself can come from many different avenues including consensus algorithm scaling, distributed ledger technology scaling, and off-chain scaling. In this report, I chose to focus on latency scaling, and identified different parallel solutions to solve the problem without compromising security or decentralization.

7 Conclusion

The fundamental aspects of what provides value in a blockchain lie in the trilemma of security, decentralization, and scalability. Security is what keeps the assets in the blockchain safe for all users. Decentralization remove a central authority as is the main idea of the Nakamoto Consensus which started the movement of decentralized blockchains. Scalability can be viewed as the user interface of a blockchain. The faster the blockchain, the more applications that can live in the blockchain and the more adoption it will have. Furthermore, a more scalable blockchain is one where the barrier to entry is very low as the gas fee is very low and that itself is decentralization. The trilemma is very connected and optimizing for all aspects without compromise makes a better overall decentralized blockchain.

Smart Contracts, which were proposed by Nick Szabo, are digital contracts that are stored on the blockchain. The contracts automate an agreement between two parties. The current technology of smart contracts in the Ethereum blockchain does not allow the execution of multiple contracts and thus slowing down the blockchain. In this paper, I proposed two approaches that optimize the speed of execution of smart contracts by taking advantage of the multiple threads in the validators to concurrently verify contracts. In the first approach, I identified a system that requires a time-based element in which the requests are sorted. The sorter, before the execution of the contracts, identifies the different types of smart contracts required and sorts them to whichever ones are not dependent on each other. The ones that are not dependent on each other are then executed in parallel, while dependent ones follow the old synchronous structure. Finally, a concept taken from the Linux kernel known as sloppy counters can be used to further speed up the blockchain. If validators create temporary state storage and then the global state is updated in each read function to the protocol, all write functions may be executed in parallel. Statistical analysis can be done on each of the methods proposed as a means of past usage of the blockchain, which can determine which method is complementary for the fastest user experience on any decentralized blockchain.

References

[1] Cloud spanner: Truetime and external consistency nbsp;—nbsp; google cloud.

[2] Mustafa Al-Bassam. Lazyledger: A distributed data availability ledger with client-side smart contracts, Jun 2019.

[3] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities, May 2019.

[4] Silas Boyd-Wickizer. An analysis of linux scalability to many cores.

[5] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform., 2014.

[6] Christophe Cerin. Methods for partitioning data to improve parallel execution time for sorting on heterogeneous clusters.

[7] Alaa Ismail El-Nashar. To parallelize or not to parallelize, speed up issue, 2011.

[8] David K Gifford. Information storage in a cecentralized computer system, 1981.

[9] Jae Kwon. Tendermint: Consensus without minting, 2014.

[10] Barbara Liskov. Practical uses of synchronized clocks in distributed systems.

[11] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Oct 2008.

[12] Fernando Pedone. Solving agreement problems with weak ordering oracles.

[13] Dr.Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2022.

[14] Anatoly Yakovenko. Solana: A new architecture for a high performance blockchain, Nov 2017.

[2] [3] [4] [5] [6] [7] [8] [1] [9] [10] [11] [12] [13] [14]