# Enabling MapReduce based Parallel Computation in Smart Contracts

[1]Yash Muchhala, [2]Harshit Singhania, [3]Sahil Sheth, [4]Kailas Devadkar
*[1,2,3,4]Department of Information Technology, Sardar Patel Institute of Technology, Mumbai, India*
*[1]yash.muchhala@spit.ac.in, [2]harshit.singhania@spit.ac.in, [3]sahil.sheth@spit.ac.in, [4]kailas_devadkar@spit.ac.in*

***Abstract*—Smart Contracts based cryptocurrencies such as Ethereum are becoming increasingly popular in various domains: but with this increase in popularity comes a significant decrease in throughput and efficiency. Smart Contracts are executed by every miner in the system serially without any parallelism, both inter and intra-Smart Contracts. Such a serial execution inhibits the scalability required to obtain extremely high throughput pertaining to computationally intensive tasks deployed with such Smart Contracts. While significant advancements have been made in the field of concurrency, from GPU architectures that enable massively parallel computation to tools such as MapReduce that distributed computing to several nodes connected in the system to achieve higher performance in distributed systems, none are incorporated in blockchain-based distributed computing. The team proposes a novel blockchain that allows public nodes in a permission-independent blockchain to deploy and run Smart Contracts that provide concurrency-related functionalities within the Smart Contract framework. In this paper, the researchers present "ConCurrency," a blockchain network capable of handling big data-based computations. The technique is based on currently used distributed system paradigms, such as MapReduce, while also allowing for fundamental parallelly computable problems. Concurrency is achieved using a sharding protocol incorporated with consensus mechanisms to ensure high scalability, high reliability, and better efficiency. A detailed methodology and a comprehensive analysis of the proposed blockchain further indicate a significant increase in throughput for parallelly computable tasks, as detailed in this paper.**

***Index Terms*—blockchain, MapReduce, concurrency, Big Data, distributed systems, security**

## I. Introduction

An eruption in Machine Learning research has led data to be one of the essential resources for advancing technology. The increase in the demand for data has consequently increased the amount of data being generated every day. Big Data, data generally more than several terabytes in size, is now a prime domain for research and advancement as it deals with processing and analyzing the vast amount of data required for Deep Learning. Hadoop, more specifically, MapReduce, is one of the most widely used Big Data processing paradigms. However, most of the big data processing is limited to data centers, which can only practically be accessible to large corporations. Individuals or small entities with a vast data influx need a more public approach towards this.

Blockchains, one of the other significant applications of Distributed Systems, is also a rapidly transitioning technology. It enables security without a need for a centralized architecture. Newer blockchains also extend their functionalities to incorporate executing code on the network instead of solely being a tradable cryptocurrency. Such executable code on the network is commonly known as Smart Contracts. They can be used to run logical codes on the nodes of the network and provide a way to compute in a completely decentralized manner. Decentralized Apps (DApps) are now commonplace in the blockchain ecosystem, which takes the general notion away from applications being client-server architectures to introduce a decentralized, open, and secure manner of running server-side code for any client application. However, Smart Contracts are generally only associated with being able to do straightforward tasks in a very secure manner. Smart Contracts are inefficient for computation that can be done in a parallel manner. Current smart contract solutions do not enable developers to code intra-smart contract parallelism that exploits the power of a large number of nodes available in the blockchain network. Enabling concurrency functionalities within Smart Contracts would help computationally intensive contracts to run parallelly at scale.

Integrating the benefits of MapReduce and the decentralized security of blockchains would theoretically result in a significant data infrastructure that is not only efficient than traditional single machine computation but also more secure, public, and decentralized. In this paper, the researchers present a proof of concept for the idea mentioned above after developing a prototype blockchain that incorporates the MapReduce paradigm inside its Smart Contract structures. The team has integrated the MapReduce paradigm with the help of a sharding protocol detailed later in this paper. A sharding protocol enables nodes in the decentralized network to simultaneously work on different parts of the data, thus enabling concurrent computation of data within contracts. The communication layer uses a publisher/subscriber architecture to interact between different nodes in the network. Although not very scalable at this point, the communication architecture is only to enable the proof of concept concurrent computation.

In implementing a functional blockchain as much as possible, the research work includes developing a rudimentary instruction set language for Smart Contracts, a randomized sharding protocol to distributed nodes into shards in a secure manner, and incorporating Proof of Work and Proof of Stake based consensus algorithms to maintain the integrity of every participating node in the network. Results obtained as part of this research work have been mentioned where the team has discussed the performance of the functioning prototype on various classic MapReduce problems. An in-depth explanation of the architecture, the methodology, and the results have been detailed further. Next, the paper talks about the different related published works around this topic and how the team reviewed existing solutions and built the "ConCurrency" network on top of them.

## II. Literature Review

### A. Fundamental Blockchain Concepts

Bitcoin [1] introduced the first blockchain introducing the Proof of Work consensus algorithm, wherein miners need to find a hash to validate the blockchain transactions. Such a consensus algorithm ensured a requirement of practically impossible computation power to pose a potential threat to the blockchain's security, but this security came at the cost of scalability. Due to the serial nature of blockchain, the mining process becomes increasingly computationally expensive and inhibits scalability.

Ethereum [2], another cryptocurrency, also based on the PoW consensus algorithm, was the first blockchain to expand its usability into a distributed computing paradigm introducing Smart Contracts. Smart Contracts is a way to run code on the distributed blockchain network nodes using the Ethereum Virtual Machine. All participating nodes in the network run all the smart contracts, though the next version of Ethereum is set to have sharding that would help offset the scalability issues that come with bitcoin fundamentals.

Loi Luu et al. [3] describe in detail a sharding protocol for permissionless blockchains; it explores and defines the failure modes and vulnerabilities of sharding as well. It also improves over existing consensus protocols in terms of efficiency. In another work by Vitalik Buterin et al. [4], they introduce a novel consensus protocol called the Proof of Stake, which is said to be included in the next version of Ethereum. It is a more scalable and less resource wasteful protocol as compared to PoW. In Proof of Stake, the authors mention that instead of mining for a hash value, the blockchain's validator nodes will have to stake their ownership of the cryptocurrency to validate blocks on the blockchain. In PoS based cryptocurrencies, the proceeding block maker is chosen via various permutations of arbitrary selections keeping into account its stake in the blockchain.

Ewa Syta et al. [5] proposed a modified version of Byzantine Fault Tolerance [6] for publicly verifiable, unbiasedly, and unpredictable randomness to divide a set of untrusted random servers into groups for scalability. It introduces RandHound and RandHerd protocols that can generate fresh random outputs after every 6 seconds while operating at a failure probability of 0.08% at the most. It provides a security mechanism for Byzantine Fault Tolerance in sharding protocols.

### B. Scalability in Blockchain-based Distributed Systems

Analyzing blockchain's scalability has been a challenging task due to the growth in blockchain technology over the past decade, as mentioned by S. Bragagnolo et al. [7] in their paper. While arguing for a novel treatment regarding big data applications with blockchain, it explores the application of parallelization techniques such as MapReduce from big data platforms to analyze blockchains that contain data worth more than 100s of gigabytes in size.

In a research work by Bartoletti et al. [8], the paper provides a way for miners to execute multiple transactions (from different smart contracts) simultaneously. Miners are expected to compute an order for the transactions based on reading/write keys first and then execute them, exploiting any possible parallelism. It uses techniques from concurrency theory as a foundation to develop truly parallel execution of transactions in a way that can be easily adapted by Ethereum and does not require a soft fork.

### C. Incorporating Blockchains with Parallelism and Concurrency

T. Dickerson et al. [9] present a novel way to permit block validators to concurrently execute independent Smart Contracts instead of the traditional serial manner that allows the miners to exploit the complete parallelization capabilities of their machines. As mentioned above, their contract is based on techniques adapted from software transactional memory [10] that allows concurrent execution 3 of non-conflicting smart contracts and determining a serializable concurrent schedule for transactions on a block. Another parallel smart contract model by W. Yu et al. [11] uses a multi-threaded approach to implement the execution of transactions in parallel. The paper also proposes a transaction splitting algorithm to avoid any synchronization problems along with a detailed analysis that shows a significant improvement in the efficiency of transaction processing.

ParBlockchain [12] introduces OXII, which is a protocol for permissioned blockchains to enable the distributed applications to execute parallelly. ParBlockchain uses the OXII as mentioned above protocol to overcome the limitations in traditional blockchain paradigms related to concurrent execution of conflicting transactions. It is designed to work with transactions with some degree of contention for concurrent computation.

Other approaches for concurrency in blockchain-based distributed systems have been surveyed in this paper by L. Gudgeon et al. [13] about off-chain computation. This paper provides a survey of layer two scaling solutions for blockchain protocols. Layer 2 or off-chain scaling involves nodes that do not communicate directly on the main chain but through some other protocol and then interact with the main chain to store results. The main chain is used only to achieve consensus for computationally light loads; side chains work on more expensive computation.

Finally, following a trend in multi-chain solutions, Plasma [14] provides an off-chain scaling solution. Private blockchains can be created by a smart contract residing on the main chain. This contract manages all of these sub-chains, and these sub-chains can compute on data parallelly, and the overall system can provide an alternative to traditional server farms.

While most of the existing research work on concurrency in the blockchain system deals with concurrently executing multiple Smart Contracts on the same node for increasing efficiency, there is hardly any research on incorporating asynchronous or concurrent code logic inside of the Smart Contract code that would enable the Smart Contract itself to be broken down into logic that would run parallelly at scale. After surveying a multitude of research papers, the team solidified the prerequisites for researching in this domain. Next, the paper discusses the methodology that their research follows and explains in-depth their proposed architecture.

## III. Methodology

ConCurrency is structured into several major components, namely the mining algorithm using Proof of Work consensus protocol, a Proof of Stake validation mechanism for verifying the results from individual shards after either the Mapper, Shuffler, or Reducer tasks that also ensures a 2/3rd majority, a publisher/subscriber communication architecture for intra-shard communication and also inter-shard communication, a randomized sharding protocol to create a group of nodes with secure randomness, and a code execution structure that would also pool the results from each of the shards in a decentralized manner.

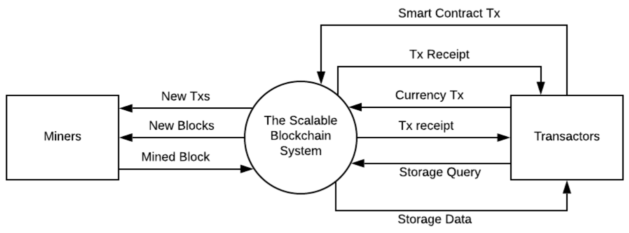

Fig. 1. Level 1 Flow Diagram for the Data Flow System

Mining using the Proof of Work protocol is done in a background thread. All the transactions received are stored in a transaction pool (tx pool). This tx pool is checked periodically by the background thread to fetch new transactions. In a case where the background thread finds new transactions, these new transactions are included in a block, and the nonce iteration begins. A nonce field is included with every block; it is incremented until the SHA3 hash of the entire block is less than a threshold determined by the blockchain network's difficulty. Every newly received block is hashed, and it is only accepted if it is less than the threshold. When a suitable nonce value is found, the block is broadcasted to the network.

Validators are registered using the Proof of Stake consensus protocol. It involves sending a predefined amount of currency to a particular address in order to register as a validator. A transactor and validator are precisely identical and will be used interchangeably in this paper for semantic purposes. Once registered as a validator, a node can execute smart contracts. A validator is grouped into shards based on its randomly generated private key. State transition transactions are only accepted if more than two-thirds of the validators assigned to that shard also have sent the transaction. Recursively applying all transactions included in the parent blockchain is the only way to get the state up to the block.

Unlike Bitcoin, and similar to Ethereum, ConCurrency uses an account-based model. Mining is strictly performed only on the highest block to maintain integrity. All the currency transactions are only accepted by all nodes in the network only if signed using the sender's private key.

### A. Architectural Overview

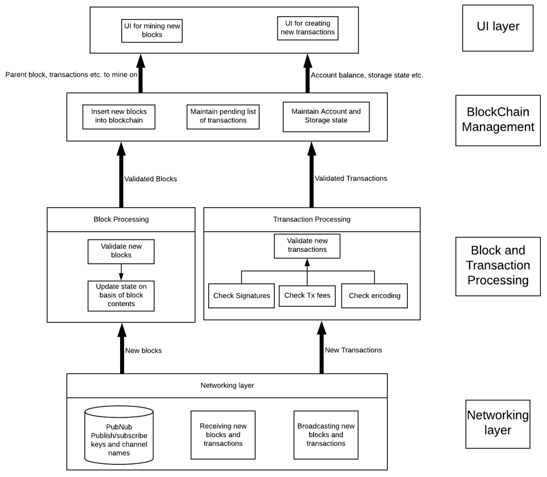
UI for mining new blocks
UI for creating new transactions
UI layer
Parent block, transactions etc. to mine on
Account balance, storage state etc.
Insert new blocks into blockchain
Maintain pending list of transactions
Maintain Account and Storage state
BlockChain Management
Validated Blocks
Validated Transactions
Block Processing
Validate new blocks
Update state on basis of block contents
Trransaction Processing
Validate new transactions
Check Signatures
Check Tx fees
Check encoding
Block and Transaction Processing
New blocks
New Transactions
Networking layer
PubNub Publish/subscribe keys and channel names
Receiving new blocks and transactions
Broadcasting new blocks and transactions
Networking layer

Fig. 2. Architecture Diagram of the Blockchain Management Component

As shown in 2, the management component comprises of four major parts: (A) Networking Layer, (B) Block and Transaction Processing, (C) Blockchain Management, and (D) Interface Layer.

*1) Networking Layer:* It is the lowermost layer of the network that handles the low-level networking-related operations. The networking layer manages the peer to peer connection between the nodes. Broadcasted transactions and broadcasted blocks are encoded and decoded in this layer. ConCurrency uses a Pub/Sub communication model where different operations are handled on different channels. All nodes in the network subscribe to particular channels in the layer and also broadcast their own transactions and blocks to the layer in order for other nodes to receive them. Due to limitations in the implementation scope, the prototype model uses the PubNub API [15] for the Pub/Sub communication method. Consequently, a full-scale network would require a custom implementation of this layer.

*2) Block and Transaction Management Layer:* Every received block and transaction are checked for their validity. All the received blocks and transactions are received from the networking layer. This layer also checks the validity of received blocks by verifying the block headers. Headers of the blockchain contain its timestamp, nonce, signatures, et cetera. Received transactions are later once again checked for their validity before being added to the tx pool by verifying the signatures with which the transactions are signed, the account balances, et cetera.

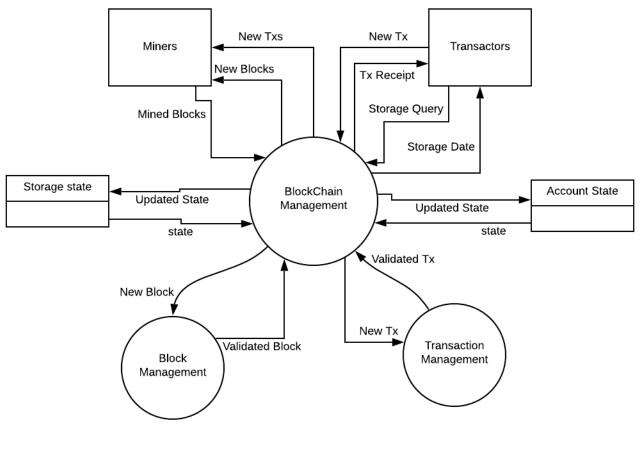

Fig. 3. Level 2 Flow Diagram for the Blockchain Management Layer

*3) Blockchain Management Layer:* The purpose of this layer is to maintain the state of the blockchain. All the received and validated blocks will be added to the chain. As described previously in the paper, the blockchain management layer maintains the contracts' account state and storage state. It also maintains a pending list of transactions so that newly mined blocks can optimize the on-boarding process for newly joined nodes.

*4) Interface Layer:* In the blockchain interface layer, the API and templates visible to the user are the main components. This layer calls the functions of the blockchain management layer using HTTP(S) operations. The mining interface calls the blockchain management layer to receive the new block's headers and the list of transactions to be included in it. The transaction interface is responsible for calling the management layer to receive the account balance, account signatures, and transaction details. Finally, it also has to call the lower layers to broadcast the created transaction. The topmost layer, i.e., the Interface layer, acts as a wrapper for the entire blockchain. In ConCurrency's implementation, the team has used FastAPI, a Python-based ASGI server framework. It uses a RESTful architecture to fulfill all the functionalities in a highly efficient manner. A complete implementation of the same can be found in the project's GitHub repository [16].

## B. Sequence Flow and Design

In this section of the paper, a flow of a node's inner workings and how miners and validators interact with the network is described with diagrammatic representations.

Above figure 4 is a sequence diagram showing how a miner will interact with the system. Figure 5 is the flowchart for the system. Firstly to mine a new block, the node must have an up to date list of transactions and blocks. In order for the node to acquire the latest list of transactions, it will request it by placing a request on the communication channel. The node will then mine a new block by calculating its headers and finally executing all the node's transactions. Once the block is mined, it is checked for validity by the other miners. If the

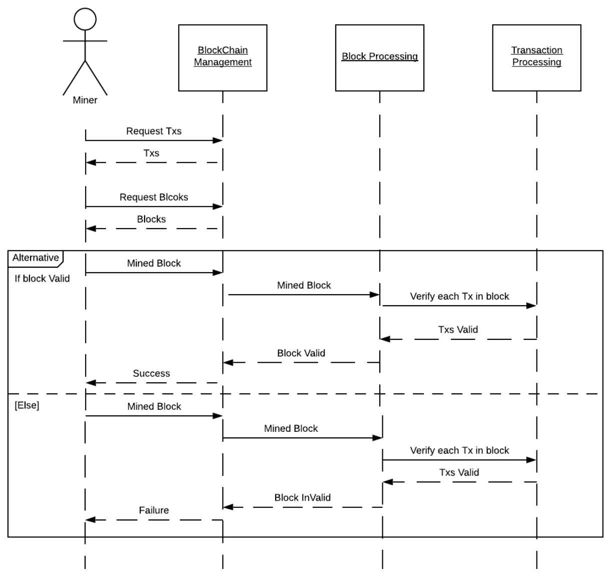

Fig. 4. Sequence Diagram Showing the Interaction of a Miner with the Network

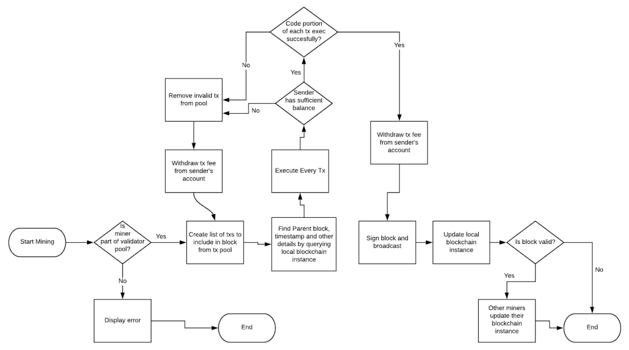
Code portion of each tx exec succesfully?
No
Yes
Yes
Remove invalid tx from pool
Sender has sufficient balance
No
Withdraw tx fee from sender's account
Withdraw tx fee from sender's account
Execute Every Tx
Start Mining
Is miner part of validator pool?
Yes
Create list of txs to include in block from tx pool
Find Parent block, timestamp and other details by querying local blockchain instance
Sign block and broadcast
Update local blockchain instance
Is block valid?
No
Yes
No
Display error
End
Other miners update their blockchain instance
End

Fig. 5. Flowchart of the ConCurrency Blockchain Network

block under consideration is deemed valid, then the block will be included in the blockchain network, or else, it is discarded in case of a rejection.

## C. Incorporating MapReduce in Smart Contracts

Until now, the paper talked about the underlying blockchain structure of the ConCurrency network. In this sub-section, the method in which a MapReduce support structure is incorporated is detailed. As mentioned previously, all communication between nodes in the network takes place through transactions. In order to identify different types of transactions, the *to_val* field is used as a key. A typical transaction contains the following fields:-

- Transaction hash: The SHA256 [10] hash of the information in the transaction.
- From Key: The initiator or sender of the transaction.
- To (*to_val*) key: A key to the transaction, this is different for different types of transactions. The list below describes what the different transactions *to_val* keys denote.

  - Transfer Currency: A transfer transaction is the simplest form of transaction. It denotes the transfer of currency from one account to another. It needs to be signed by the sender's private key, and the *to_val* key is the address of the receiver's account.
  - Register Validator (VAL): Whenever a node wants to be registered as a validator, it would broadcast a block carrying this transaction. It also needs to be signed by the private key of the originating node's account, but the *to_val* key is the literal string "VAL" denoting it as a validator registration transaction.
  - Deploy Contract (DEPLOY): A deploy transaction broadcasts the base64 encoded Mapper, Reducer, and Shuffler files in order for all the validator nodes to get them. Once a "DEPLOY" transaction is in the blockchain, the contract code remains on the blockchain and can be "RUN" anytime.
  - Run Contract (RUN): To run a deployed contract, a "RUN" transaction needs to be broadcasted in a block from the node in the network; upon procurement of this transaction from other nodes in the network, they are allocated into shards, and each of the nodes executes the smart contracts as defined in the deployment according to the index of shard the node belongs to.
  - New State: During the execution of a contract, whenever a state transition occurs in a shard, for example, a state transition from the mapper function execution to the shuffler function execution, a transaction with the new state is broadcasted in the network.
- Mapper Code: An optional field for the base64 encoded python code for the mapper. It is only mandatory in case of a "DEPLOY" transaction type.
- Reducer Code: An optional field for the base64 encoded python code for the reducer. It is only mandatory in case of a "DEPLOY" transaction type.
- Shuffler Code: An optional field for the base64 encoded python code for the shuffler. It is only mandatory in case of a "DEPLOY" transaction type.
- Number of Shards: An optional field to specify the number of shards for a MapReduce operation. It is only mandatory in case of a "DEPLOY" transaction type.

A shard is a group of validator nodes in the network that have been grouped in an anonymous manner such that they only know the shard they belong to and are completely unaware of the location of nodes in their shard or any other shard. When the "DEPLOY" transaction is broadcasted, all the validator nodes receive the transaction along with the base64 encoded mapper, shuffler, and reducer codes. They remain in the blockchain until the end of the blockchain's lifetime, i.e., until all nodes shut down. Now, if a certain node wants to run the contract, it specifies the transaction hash along with broadcasting the transaction of type "RUN." Once a node specifies the "RUN" transaction, the following steps take place in order to compute a MapReduce operation securely:-

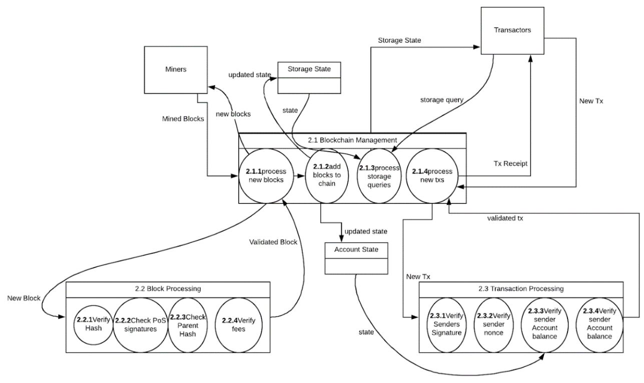

Fig. 6. Level 3 Flow Diagram for the Block Transaction Processes

1) A shard number for each node is calculated using pseudo-random numbers. The current timestamp is used as the seed to the pseudo-random number generator, and then a modulo operation is used with the generated pseudo-random number, the private key of the node, and the total number of shards for the entire MapReduce operation to determine the shard number of the current node. In this way, the researchers have devised a completely safe and decentralized way of sharding nodes in the network such that each node is only aware of its shard number. Any node in the network is unaware of any other node's shard indices, thus maintaining anonymity.
2) Once all individual nodes calculate their shard number, all the base64 encoded codes are decoded and saved in memory.
3) Execution of the mapper function in each node begins, and every shard runs its mapper code according to the part of the data provided in the mapper class.
4) Once the nodes in a shard complete their execution in the mapper phase, they then wait for all shards to complete their respective mapper executions. Once all shards broadcast the *New State* transaction, each node will then stop its waiting stage and execute the shuffler function.
5) Like the mapper stage, the execution of the shuffler nodes will take place in this stage, and the nodes that complete before the other nodes go into a waiting state. As soon as all nodes are done executing the shuffler function*, i.e., the nodes receive the broadcast for *New State* transaction from a two-thirds majority of the nodes in the shards, the MapReduce operation comes to its final state.
6) In the final stage, the reducer codes are executed, and the output from each of the shards is reduced to a singular output, which is the final output. A complete execution log is generated at the end of each successful reducer output. Each node can query the state of this "RUN" contract to know whether the MapReduce execution is

complete or not.

*Although it is mentioned that all nodes wait for all nodes to finish their execution, it is mentioned so just for clarity of explanation. In reality, for fault tolerance purposes, ConCurrency nodes only wait for a two-thirds majority of the nodes to end their waiting stage. All data communication is done in small chunks using the same pub/sub communication layer to maintain efficiency.

The above steps cover the entire flow of the research's implementation of the Smart Contract System's MapReduce structure. Using this implementation, the team has analyzed several classical MapReduce problems in order to test the prototype rigorously. The next section details the findings obtained from testing the prototype implementation.

## IV. Results

In this section, the researchers present a detailed analysis of their tests. They tested their prototype on three different MapReduce operations: (A) letter frequency count using MapReduce, (B) finding the number of primes numbers in a range, and (C) an implementation of Naive Baye's for Machine Learning. Due to limitations in resources, a maximum of only 12 validator nodes has been used for testing.

### A. Letter Frequency Count using MapReduce

A classical MapReduce problem is implemented in this test, i.e., counting the letters' frequency in a document. To demonstrate a fair Big Data computation, this test uses a document with over 100 million letters. Below in fig. 7 are the results for each of them in terms of runtime per number of shards. The runtime to the number of shards follows a linear curve. This is the expected behavior as the letter count functions in each of the mapper, shuffler, and reducer stage will take the same amount of time for any input. Next, an example where this linear pattern is not observed is demonstrated.

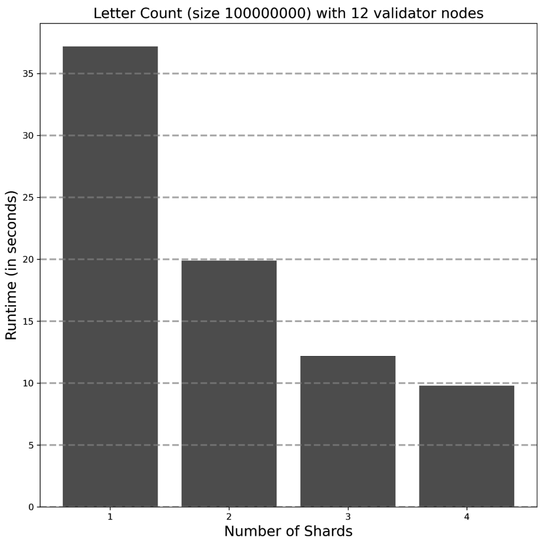

Fig. 7. Results for the Letter Frequency Count Operation

### B. Finding Primes

For the test involving finding the number of primes till N, it is not expected for the runtime to number of shards graph to follow a linear pattern. Since the algorithm uses O($n^2$) time complexity, any computation from 1 to N / 2 would take proportionally less amount of time as compared to the computation from N / 2 to N, This is observed in both of the tests below, i.e. for N=96,000 in fig. 8 and also, for N=192,000 in fig. 9. As expected, a linear ratio is not followed in either of the graphs for the finding primes function. The graph instead follows the curve in accordance with the time complexity of the underlying algorithm.

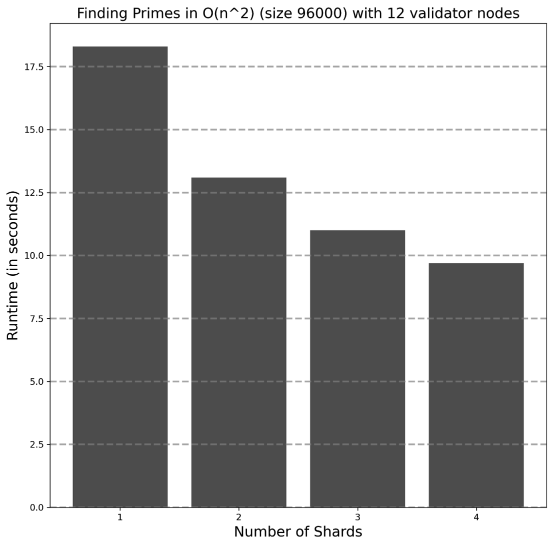

Fig. 8. Results for the Counting Primes function for N=96,000

### C. Training a model with the Naive Baye's Algorithm

A novel mapper, reducer, and shuffler methods for Naive Baye's algorithm suitable for MapReduce in ConCurrency were the inputs for this test case, along with a large dataset. Below, in fig. 10, are the training execution results in terms of runtime and number of shards. Similar to the letter frequency example, the Naive Baye's model also follows a linear pattern with negligible overhead. This concludes the results section of the paper, and the future scope of the project is discussed in the next section.

## V. Conclusion and Future Work

In conclusion, the research team has successfully demonstrated a prototype for a MapReduce-based concurrency within Smart Contracts that could potentially enable Big Data computations to be executed directly on the blockchain. ConCurrency

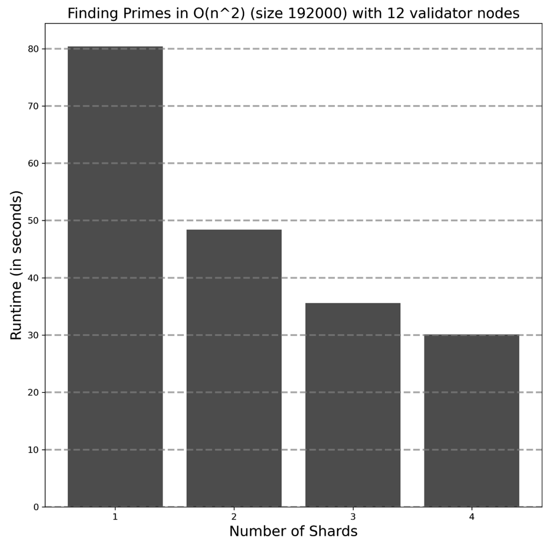

Fig. 9. Results for the Counting Primes function for N=192,000

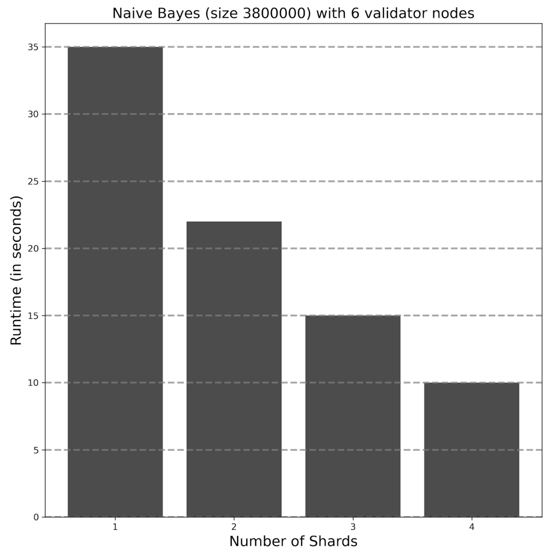

Fig. 10. Results for the Training of a Machine Learning model using the Naive Baye's Algorithm

has shown promising results in terms of maintaining cryptographic security alongside computing MapReduce-based Big Data tasks concurrently.

As future research work, the team expects to incorporate a complete instruction set language on top of the existing rudimentary one for Smart Contracts to be natively compiled. Further, the researchers plan to stress-test the network to identify any weak spots upon access to more resources. Finally, to create a fully-functional blockchain network, there is a need to device a distributed file system to store such big data.

ConCurrency, with this research paper, although with its shortcomings, provides an exhaustive proof of concept for processing big data using blockchain security.

## REFERENCES

[1] Bitcoin: A Peer-to-Peer Electronic Cash System: Available: https://bitcoin.org/bitcoin.pdf
[2] Ethereum Whitepaper: Available:https://ethereum.org/whitepaper/
[3] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A Secure Sharding Protocol For Open Blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS' 16). Association for Computing Machinery, New York, NY, USA, 17–30
[4] Proof of stake: Available: https://arxiv.org/abs/1710.09437
[5] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine fault tolerance. In Proceedings of the third symposium on Operating systems design and implementation (OSDI' 99). USENIX Association, USA, 173–186
[6] Ewa Syta and Philipp Jovanovic and Eleftherios Kokoris Kogias and Nicolas Gailly and Linus Gasser and Ismail Khoffi and Michael J. Fischer and Bryan Ford. Scalable Bias-Resistant Distributed Randomness, 2016
[7] Santiago Bragagnolo, Matteo Marra, Guillermo Polito, and Elisa Gonzalez Boix. 2019. Towards scalable blockchain analysis. In Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB' 19). IEEE Press, 1–7
[8] Bartoletti M., Galletta L., Murgia M. (2020) A True Concurrent Model of Smart Contracts Executions. In: Bliudze S., Bocchi L. (eds) Coordination Models and Languages. COORDINATION 2020. Lecture Notes in Computer Science, vol 12134. Springer, Cham
[9] Thomas Dickerson, Paul Gazzillo, Maurice Herlihy, and Eric Koskinen. 2017. Adding Concurrency to Smart Contracts. In Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC' 17). Association for Computing Machinery, New York, NY, USA, 303–312.
[10] Nir Shavit and Dan Touitou. 1995. Software transactional memory. In Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing (PODC' 95). Association for Computing Machinery, New York, NY, USA, 204–213.
[11] Yu, Wei & Luo, Kan & Ding, Yi & You, Guang & Hu, Kai. (2018). A Parallel Smart Contract Model. MLMI2018: Proceedings of the 2018 International Conference on Machine Learning and Machine Intelligence. 72-77. 10.1145/3278312.3278321.
[12] M. J. Amiri, D. Agrawal and A. El Abbadi, "ParBlockchain: Leveraging Transaction Parallelism in Permissioned Blockchain Systems," 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, 2019, pp. 1337-1347, doi: 10.1109/ICDCS.2019.00134.
[13] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. SoK: Layer-Two Blockchain Protocols, 2019
[14] Poon, Joseph and Buterin, Vitalik. Plasma: Scalable autonomous smart contracts, 2017
[15] "Realtime Communication APIs for Chat, Notifications, Geolocation, and Collaboration," PubNub, 21-Aug-2020. [Online]. Available: https://www.pubnub.com/. [Accessed: 31-Oct-2020].
[16] Github repository. [Online]. Available: https://github.com/yashmuchhala/ConCurrency.