Turnitin Originality Report

VAR by Aarti Dinkar

From ym4 (Student)

- Processed on 21-May-2019 14:14 IST
- ID: 1133789225
- Word Count: 7761

Similarity Index
20%
Similarity by Source

Internet Sources:
18%
Publications:
12%
Student Papers:
N/A

07-May-2019].

[13] "Go Ethereum." Available: https://geth.ethereum.org/. [Accessed: 01-May-2019].

[14] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, "Demystifying Incentives in the Consensus Computer," pp. 706–719, 2015.

[15] J. Osterrieder, S. Chan, J. Chu, and S. Nadarajah, "A Statistical Analysis of Cryptocurrencies," 2017.

[16] "Solidity." Available: https://solidity.readthedocs.io/en/v0.5.3/. [Accessed: 01-May-2019].

[17] M. Alharby and A. van Moorsel, "Blockchain-based Smart Contracts: A Systematic Mapping Study," pp. 125–140, 2017.

[18] T. Dickerson, P. Gazzillo, M. Herlihy, and E. Koskinen, "Adding Concurrency to Smart Contracts," 2017.

[19] E. Hildenbrandt, M. Saxena, X. Zhu ECNU, N. Rodrigues, and P. Daian, "KEVM: A Complete Semantics of the Ethereum Virtual Machine," pp. 1–33, 2017.

[20] S. Tikhomirov, "Ethereum: State of knowledge and research perspectives," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10723 LNCS, pp. 206–221, 2018.

[21] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: An efficient multithreaded runtime system," *J. Parallel Distrib. Comput.*, vol. 37, no. 1, pp. 55–69, 1996.

[22] Schoenmakers, B., & Tuyls, P. (2006, May). Efficient binary conversion for Paillier encrypted values. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (pp. 522-537). Springer, Berlin, Heidelberg.

[23] "Truffle Framework." Available: https://truffleframework.com/. Accessed: 01-May-2019].

[24] https://www.ethervm.io/. Accessed: 01-May-2019

References

[1] A. G. Dinker and V. Sharma, "Attacks and challenges in wireless sensor networks," *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, 2016, pp. 3069-3074.

[2] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," in *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017*, 2017.

[3] M. Crosby, Nachiappan, P. Pattanayak, S. Verma, and V. Kalyanaraman, "BlockChain Technology: Beyond Bitcoin," *Appl. Innov. Rev.*, 2018.

[4] F. Li, D. Pieńkowski, A. Van Moorsel, and C. Smith, "A Holistic Framework for Trust in Online Transactions," *International Journal of Management Reviews*. 2012.

[5] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2013.

[6] G. WOOD, "ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER," *Ethereum Proj. Yellow Pap.*, 2018.

[7] F. Hisch, "Namecoin," *Netw. Archit. Serv.*, 2014.

[8] Litecoin.info, "Litecoin," *litecoin.info*, 2018.

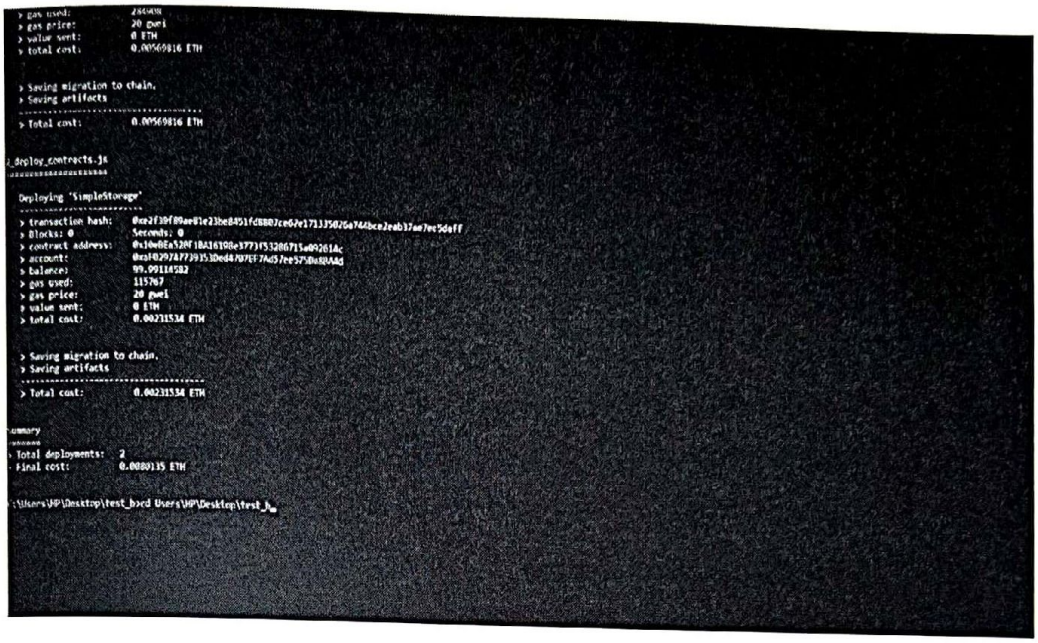
[9] Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., ... & Song, D. (2016, February). On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security* (pp. 106-125). Springer, Berlin, Heidelberg.

[10] G. Fox, "Peer-to-peer networks," *Computing in Science and Engineering*. 2001.

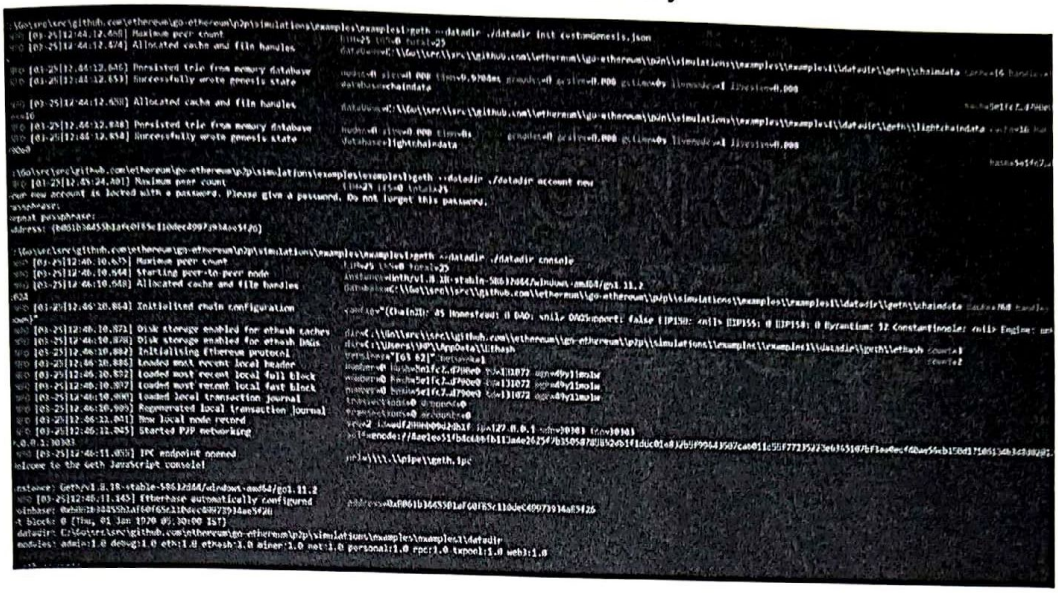
[11] M. Wohrer and U. Zdun, "Smart contracts: Security patterns in the ethereum ecosystem and solidity," *2018 IEEE 1st Int. Work. Blockchain Oriented Softw. Eng. IWBOSE 2018 - Proc.*, vol. 2018-Janua, pp. 2–8, 2018.

[12] "The Go Programming Language." Available: https://golang.org/. [Accessed:

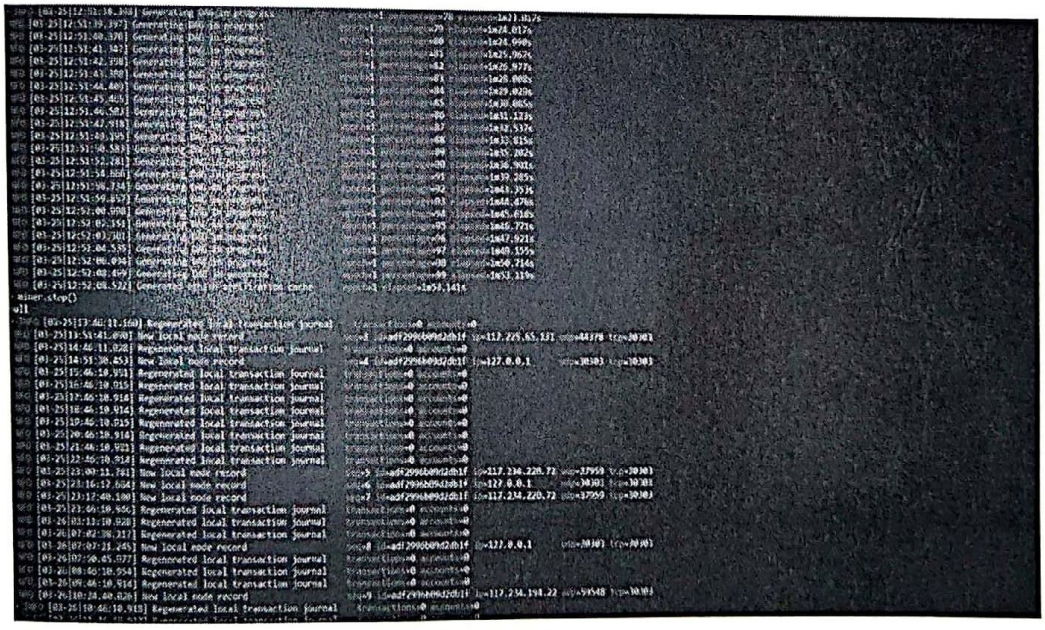
Screenshot 3: Validation of blockchain



Screenshot 1: Modification in EVM executed correctly



Screenshot 2: Mining process for a smart contract



```
        c := new(big.Int).Mod(prod, pubkey.Nsq)

        return c.Bytes(), nil
}
```

2. Instruction for Decryption in the core/vm/instructions.go file

```
func OpDecr(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack
*Stack,
privkey *PrivKey, cipher []byte) ([]byte, error) {

        c := new(big.Int).SetBytes(cipher)

        if privkey.Nsq.Cmp(c) < 1 {
                return nil, ErrLongMessage
        }
        //c^l mod n^2
        a := new(big.Int).Exp(c, privkey.L, privkey.Nsq)
        //L(x) = x-1 / n we compute L(a)
        l := new(big.Int).Div(new(big.Int).Sub(a, one), privkey.N)
        //computing m
        m := new(big.Int).Mod(new(big.Int).Mul(l, privkey.U), privkey.N)
        return m.Bytes(), nil

}
```

Appendicies

"Appendix A"

Screenshots of the implementation

1. Instruction for Encryption in the core/vm/instructions.go file

```
func OpEncr(pc *uint64, evm *EVM, contract *Contract, memory *Memory, stack
*Stack, pubkey
*PubKey, message []byte) ([]byte, error) {

        r, err := rand.Prime(rand.Reader, pubkey.KeyLen)
        if err != nil {
                return nil, err
        }
        m := stack.pop()
        m := m.SetBytes(message)
        if pubkey.N.Cmp(x) < 1 {
                return nil, ErrLongMessage
        }
        //c = g^m * r^nmod n^2

        //g^m
        gm := new(big.Int).Exp(pubkey.G, m, pubkey.Nsq)
        //r^n
        rn := new(big.Int).Exp(r, pubkey.N, pubkey.Nsq)
        //prod = g^m * r^n
        prod := new(big.Int).Mul(gm, rn)
```

CHAPTER 6

CONCLUSION

The proposed mechanism for parallel execution of opcodes increases the throughput by executing the smart contracts written in solidity on ethereum. It also improves throughput of mining and validation by executing the unrelated contracts parallelly. The results in the previous section show that the synchronous execution mechanism developed increases the speed of mining and produces fewer conflicts. It is seen that the conflicts among transactions of the same block are comparatively less in more time.

Synchronous execution of smart contract increases the speed of the mining process by enabling the miners to construct blocks quickly before adding them to the chain. But in the case of permissionless blockchains, the maximum time of the miner is spent in computing PoW for the block. The simultaneous execution of smart contracts provides various advantages to the validators. The time taken to execute the transactions by validators is much more than the time consumed in validating the PoW performed by the miners. The PoW offers much computational burden. Thus, to reduce the computational problems, ethereum is moving to PoS. Additionally, permission blockchains refrain themselves from using PoW, thus synchronous execution of smart contract provides a better throughput.

Figure 5.2 describes the contracts speedup for miners and validators based on the conflicts percentage and number of transactions being processed respectively. The miners and validators are able to process more transactions in a given time with less conflicts. Thus, the overall speed of processing blockchain gets increased to 1.61x which is higher than than the serial execution speed.

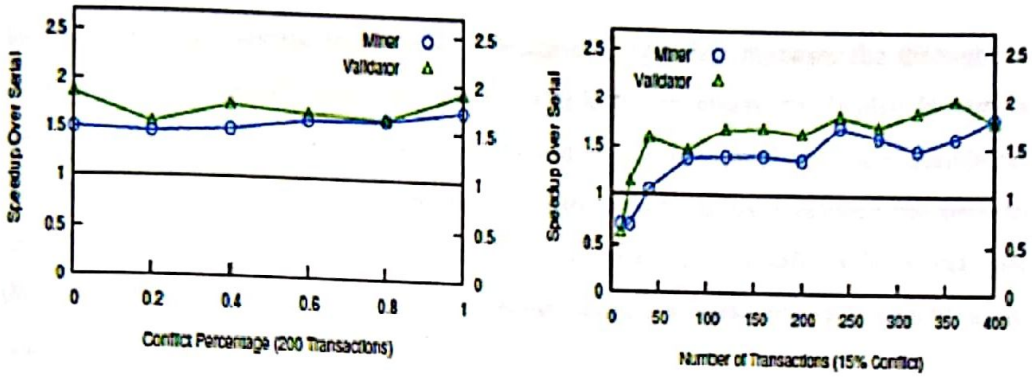


Fig. 5.2. The speedup of contracts for the miners and validators

transactions. This gives various advantages like improved throughput, resource utilization, reduced waiting time, average response time of transaction increased, etc.

Table 5.2: Comparison between EVM programming languages

PROGRAMMING LANGUAGE FOR EVM	SPEED UP FOR MINERS	SPEED UP FOR VALIDATORS	MEAN BLOCK TIME
Java	1.33x	1.69x	14-15 sec
Go Lang	1.45x	1.78x	14-15 sec

Most common platform for execution of EVM are written in Go Lang or Java. Table 5.2 describes the correlation between the platforms implemented in the EVM. From the analysis it has been found that proposed technique in Go Lang helps provide better throughput than Java based EVM. Figure 5.1 describes a graphical comparison between the serial and parallel execution mechanisms. It portrays that the parallel execution mechanism takes less time to process the smart contracts in EVM than serial execution but with using comparatively more memory.

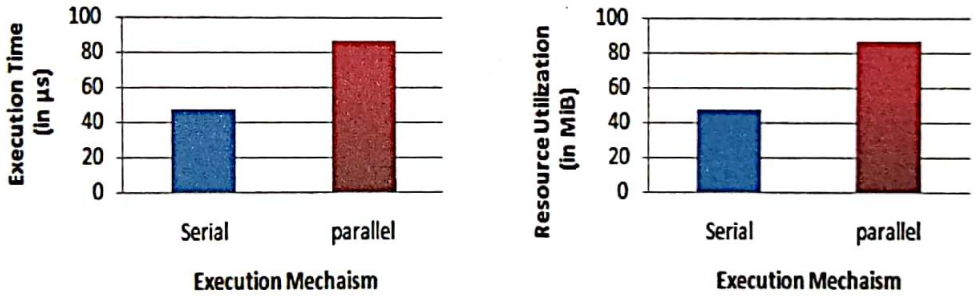


Fig. 5.1: Comparison of Serial and Parallel execution mechanism

transactions. This gives various advantages like improved throughput, resource utilization, reduced waiting time, average response time of transaction increased, etc.

Table 5.2: Comparison between EVM programming languages

PROGRAMMING LANGUAGE FOR EVM	SPEED UP FOR MINERS	SPEED UP FOR VALIDATORS	MEAN BLOCK TIME
Java	1.33x	1.69x	14-15 sec
Go Lang	1.45x	1.78x	14-15 sec

Most common platform for execution of EVM are written in Go Lang or Java. Table 5.2 describes the correlation between the platforms implemented in the EVM. From the analysis it has been found that proposed technique in Go Lang helps provide better throughput than Java based EVM. Figure 5.1 describes a graphical comparison between the serial and parallel execution mechanisms. It portrays that the parallel execution mechanism takes less time to process the smart contracts in EVM than serial execution but with using comparatively more memory.

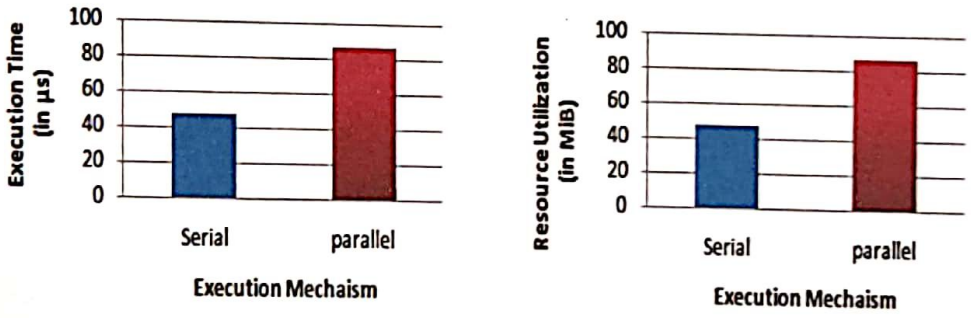


Fig. 5.1: Comparison of Serial and Parallel execution mechanism

transactions. This gives various advantages like improved throughput, resource utilization, reduced waiting time, average response time of transaction increased, etc.

Table 5.2: Comparison between EVM programming languages

PROGRAMMING LANGUAGE FOR EVM	SPEED UP FOR MINERS	SPEED UP FOR VALIDATORS	MEAN BLOCK TIME
Java	1.33x	1.69x	14-15 sec
Go Lang	1.45x	1.78x	14-15 sec

Most common platform for execution of EVM are written in Go Lang or Java. Table 5.2 describes the correlation between the platforms implemented in the EVM. From the analysis it has been found that proposed technique in Go Lang helps provide better throughput than Java based EVM. Figure 5.1 describes a graphical comparison between the serial and parallel execution mechanisms. It portrays that the parallel execution mechanism takes less time to process the smart contracts in EVM than serial execution but with using comparatively more memory.

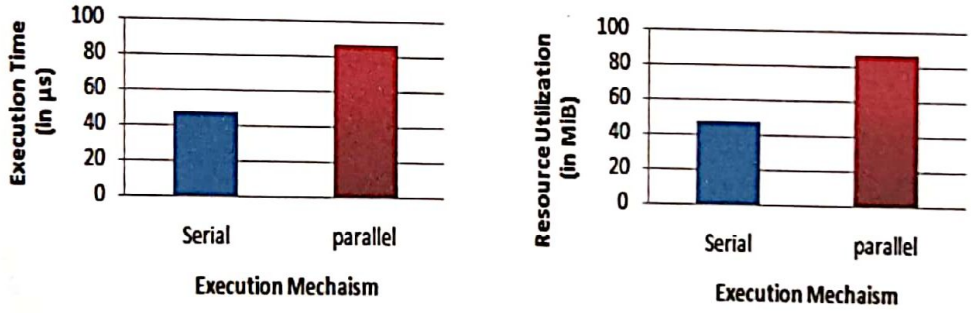


Fig. 5.1: Comparison of Serial and Parallel execution mechanism

transactions. This gives various advantages like improved throughput, resource utilization, reduced waiting time, average response time of transaction increased, etc.

Table 5.2: Comparison between EVM programming languages

PROGRAMMING LANGUAGE FOR EVM	SPEED UP FOR MINERS	SPEED UP FOR VALIDATORS	MEAN BLOCK TIME
Java	1.33x	1.69x	14-15 sec
Go Lang	1.45x	1.78x	14-15 sec

Most common platform for execution of EVM are written in Go Lang or Java. Table 5.2 describes the correlation between the platforms implemented in the EVM. From the analysis it has been found that proposed technique in Go Lang helps provide better throughput than Java based EVM. Figure 5.1 describes a graphical comparison between the serial and parallel execution mechanisms. It portrays that the parallel execution mechanism takes less time to process the smart contracts in EVM than serial execution but with using comparatively more memory.

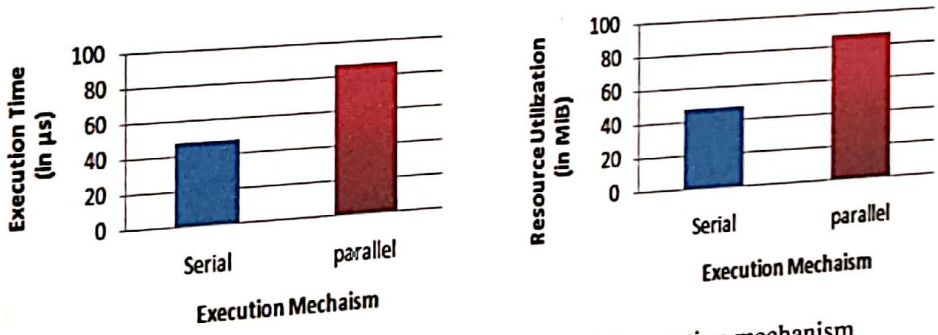


Fig. 5.1: Comparison of Serial and Parallel execution mechanism

CHAPTER 5

RESULTS AND ANALYSIS

In the previous chapter we have implemented the algorithms for parallel execution of opcodes, miners and validators.

The concurrent execution by a miner increases the possibility of high throughput. This provides an upper hand over other miners. The synchronization conflicts that occur among the smart contracts cause some contracts to be rolled back and re-executed. This causes delay in construction of a new block. Thus, the miner is forced to re-execute code without any reimbursements. However, the experimental results demonstrate that a small extent of synchronized execution improves the efficiency, even in case of moderate conflicts.

Table 5.1: Comparison between Execution mechanisms

Properties	Serial Execution	Parallel Execution
Throughput	Low	High
Resource Utilization	48 MiB	86 MiB
Waiting Time	High	Low
Execution Time	305.154 µs	236.633 µs

Executing multiple transactions simultaneously is called concurrent executions. Table 5.1 describes the connection between the various execution techniques. In spite of the fact that executing transactions sequentially involves consistency, safety and like features, but the time spent on sequential execution cannot be compromised. Thus, the multi-user systems need to implement the concept of concurrent execution of

Thus, locks are abstracted for tracking dependencies. Each lock incorporates a counter to keep a count of the number of times it was unlocked by a committing action during the development of the current block. When a miner starts a block, these counters are initialized to zero. Upon commit the counter is incremented for each lock held, and after that it enlists the lock profile with the VM recording the abstract locks and their counter values.

4.5 Summary

In this chapter, the system model has been described followed by the assumptions and notations used for describing the proposed scheme. It explains the implementation mechanism incorporated in detail.

Miner processes multiple sequential contracts parallelly. Contracts' information are instrumented to identify synchronization clashes at run-time, in the same way as mechanisms like transactional boosting. If one speculative contract execution conflicts with another, the conflict is resolved either by delaying one contract until the other completes, or by rolling back and restarting one of the conflicting executions. If the action gets completed successfully, it is said to commit, and otherwise it aborts.

Phase 3: The technique proposed for miners in phase 2, does not provide any help to the validators. This is because, the miners find a concurrent schedule for a block's transactions, which is equivalent to a sequential schedule. To check that the block's mining was correct, validators use an approach to reproduce the miner's concurrent schedule. Figure 4.5 describes the algorithm of parallel execution for validation.

Parallel Validation

Require: S and G from miner

Ensure: A set of F guaranteeing parallel execution according to G

1: **function** CONSTRUCTVALIDATION(B)
2: Initialize F from each transaction t to its fork-join task f
3: Make G′ by reversing the edges of G
4: **for all ct ∈ S do**
5: L←all transactions u ∈ G′ that happen immediately before ct
6: Create f for t that first joins with all tasks in R, i.e.,
f ← for (l in L) { F . get(l). join () } execute(ct)
7: Save the new fork-join task in F , i.e., F .put(t , f)
8: **end for**
9: **return** F
10: **end**

Fig. 4.5: Algorithm for parallel validation

numbers which would be equal to 300 in this case. Then, the whole blockchain is processed from block 0 until today, if the same block.root() is obtained at the end, this means the modification has been done correctly. The parallel execution of opcodes can be checked by testing the instruction against go-panics. This can be done, by developing a smart contract which executes the encryption and decryption of numbers. Upon execution, if the desired results are obtained, this shows that the opcodes have been implemented properly.

Phase 2: After parallel execution of opcodes, a speculative technique for mining is formulated. The semantics of smart contracts are sequential. Every miner uses single threading, as a result EVM instruction is executed one at a time. The miner executes the contracts in each block's in sequence. Contract can call other contract's functions, causing control to pass from the first contract code to the second, and back again. Thus, we can conclude that the concurrent smart contract executions use less time than sequential executions. From above we conclude that miners should execute contract codes as speculative actions. Figure 4.4 describes the algorithm of parallel execution for mining.

Parallel Mining

Require: set of CT

Ensure: G and S are present

1: **function** MINEPARALLEL(B)

2: Initialize L

3: Execute all transactions ct ∈ CT in parallel, recording activities in R

4: Generate G from R

5: Create S via sorting G

6: **return** (S,G)

7: **end**

Fig. 4.4: Algorithm for Parallel Mining

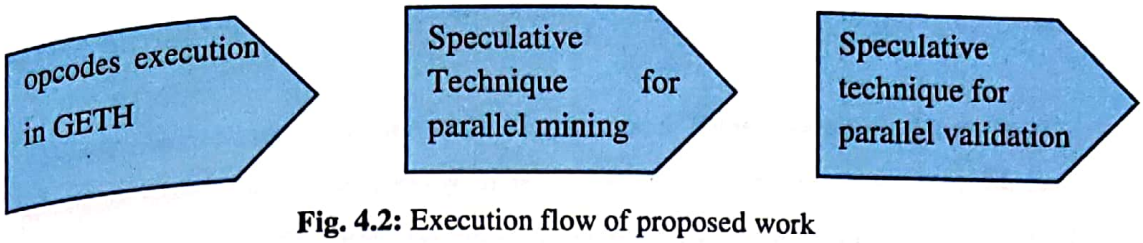


Fig. 4.2: Execution flow of proposed work

Phase 1: The EVM has defined different opcodes for different operations. To execute a transaction the GETH invokes opcodes serially in the order in which they are sequenced in the stack. For example, to add two encrypted numbers, first the opcode for encryption is called for the two numbers sequentially and then the add opcodes is called for adding the encrypted numbers. This process consumes large memory and more time, leading to less efficiency in execution of transactions. To overcome this problem, the opcodes should be executed parallelly using go-routines. The memory of EVM consists of approximately 116 undefined spaces which can be utilized for defining user-defined opcodes. These opcodes can be utilised to implement parallelism. For example, to check the parallelism paillier encryption [23] technique is incorporated as user-defined EVM opcode.

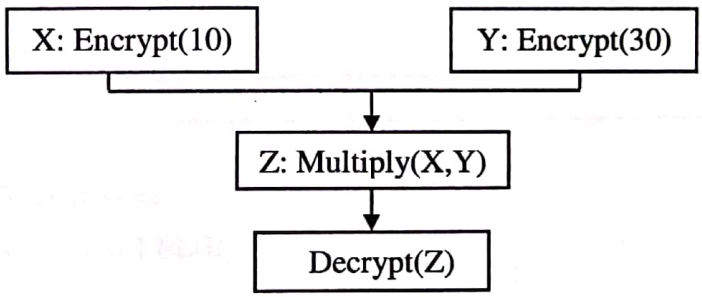


Fig. 4.3: The paillier encryption example

In Figure 4.3, an example of paillier encryption is described. There are two numbers X:=10 and Y:=30. These numbers are encrypted separately. Then, the encrypted numbers are multiplied using multiply opcode. The encrypted result is decrypted. The decrypted value should generate similar output as multiplication of the two input

4.3 Assumptions and Notations

The proposed mechanism adopts the above mentioned system model and makes the following assumptions like:

i. The blockchain should be implemented in a secured P2P network.
ii. It is assumed that all the P2P nodes have uninterrupted connectivity.
iii. The blockchain should be developed on system with high configurations.
iv. EVM code can carry out any computation, including infinite loops.

The Table 4.1 describes the notations used throughout the paper for describing the mechanism.

Table 4.1: Symbols and Notations used

Symbols	Description
CT	Contract transactions
S	Transactions in serial order
G	Happens-before graph of the Schedule
G'	Happens-after graph
L	Block of transactions
R	Log for recording operations
F	Fork-join tasks

4.4 Proposed Scheme

The work is executed in three phases: Phase 1 describes the parallel execution of opcodes written in GETH. Phase 2 and 3 describes the speculative techniques used for parallel mining and validation. In Figure 4.2, the execution flow of the proposed work has been described.

in comparison to threads since one go-routine runs on one thread at a time. The opcodes are processed concurrently in the EVM stack and hence, speedup the execution of the transactions.

Execution Model – To secure the block of transactions, complex computations need to be carried out with the help of mining and validation. In mining, one transaction is processed at one time. As a result more time is consumed and validation is delayed. Thus, to overcome this issue, miners execute the smart contracts parallelly. The non-conflicting contracts are executed simultaneously, and a synchronous schedule is created for the transactions in a block. This graph is executed by validators for re-executing the miner's parallel process simultaneously.

Following approach is followed for adding concurrency to smart contracts. Figure 4.1 illustrates an overview of the methodology process of this work which is fully explained in this chapter. The emphasis is on implementation mechanism in detail.

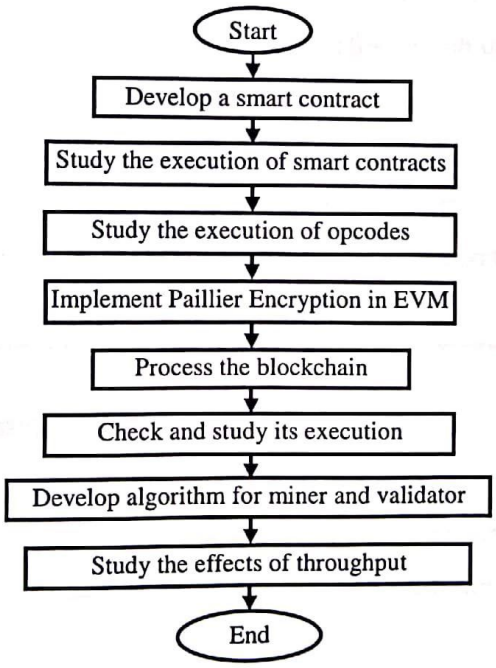


Fig. 4.1: System Model of the proposed work

CHAPTER 4

PROPOSED WORK

4.1 Overview

Our proposed scheme is focusing on the improvement in the execution speed of the miners and validators in the P2P network of the ethereum blockchain. A transaction is initiated between two parties using ethereum mist wallet. The address of the sender's wallet is linked with the private key. The transaction is then signed by the sender using his digital signature and is processed in the EVM. The EVM processes the transactions into opcodes. The opcodes are executed parallelly using go-routines. The nodes accumulate the new transactions into a block. After a block time of 14-15 sec, the miner nodes in the P2P network try to transform the state of associated contracts in the block. If a miner is successful in solving the complex computational problem, that miner broadcasts the block to all the nodes with a happen-before graph [15]. The validators use this graph to re-execute the blockchain. This is done by re-executing the miner's parallel schedule concurrently. The nodes accept the block and add it to their record. By adding parallelism to smart contract execution, better response can be achieved in less time.

4.2 System Model

The methodology for adding concurrency to smart contracts and transactions can be divided into two categories for simplification.

Language Model - The EVM processes the transaction according to the opcodes defined in it. It has large unused memory which can be used for implementing user-defined opcodes. These opcodes can be used to execute the transactions parallelly. To achieve concurrency, most languages use threads. But in GETH, the parallelism in opcodes execution can be achieved using go-routines. Go-routines are more efficient

Table 3.1 Opcodes Description

Operations	Opcodes	Gas	Description
ADD	0X01	3	Arithmetic Operations
SUB	0X02		
MUL	0X03	5	
DIV	0X04		
ADDMOD	0X08	8	
MULMOD	0X09		
AND	1x06	3	Bitwise Logic Operations
OR	1x07		
XOR	1x08		
LT	1x00	3	Comparison Operators
GT	1x01		
PUSH	6x00	3	Stack Operations
DUP	8x00		
SWAP	9x00		
POP	5x00	2	
MLOAD	5x01	3	Memory Operations
MSTORE	5x02		
JUMP	5x06	8	Unconditional Jump
SLOAD	5x04	200	Storage Operations
SSTORE	5x05	5,000 20,000	
BALANCE	3x01	400	Retrieving account's Balance
CREATE	fx00	32,000	Create new account
CALL	fx01	25,000	Create new account
SHA3	2x00	20	Hash Operation

3.3 Summary

In this chapter, we described the algorithms, technologies, and tools used for the development of the proposed work. In the next chapter, based on the methodology described above we have provided a solution to the problem encountered in the execution of smart contracts in ethereum.

Smart Contracts are executed using a specific set of instructions called opcodes. The execution of contracts starts at the beginning of the bytecode. There are currently 140 unique opcodes defined in the EVM for different operations like comparison, stack-manipulation, memory-manipulation, storage- manipulation, halting, etc. Each opcode uses memory, to hold transient data while the contract is getting executed. The EVM can have maximum of 256 opcodes. The opcodes are encoded to bytecode for storing them efficiently. Each opcode is encoded as one byte [24]. Each opcode is allocated byte, for example: add opcode is 0 x 01. The ethereum node can be used by anyone to execute smart contracts. Thus an attacker can try to slow down the network by creating contracts with high computations. To prevent the occurring of such attacks, each opcode has its associated base gas cost. Additional gas is charged for more complex opcodes. Thus, gas can be defined as a factor used for estimating the computational performance of smart contracts and transactions in the Ethereum. Figure 3.2 describes the scenario of the current memory used in EVM for opcodes. Table 3.1 describes some of the operations performed by different opcodes in EVM along with the gas consumed by each opcode.

00	01	02	03	04	05	06	07	08	09	0A	0B	-	-	-	-
10	11	12	13	14	15	16	17	18	19	1A	-	-	-	-	-
20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	-
40	41	42	43	44	45	-	-	-	-	-	-	-	-	-	-
50	51	52	53	54	55	56	57	58	59	5A	5B	-	-	-	-
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	-	-	-	-	-	-	-	-	-	-	-
B0	B1	B2	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
F0	F1	F2	F3	F4	-	-	-	-	-	FA	-	-	FD	-	FF

Fig. 3.2: Opcodes defined in the EVM

3.2.3 Geth

Go-ethereum client is commonly referred as GETH [13]. It is the official golang implementation of the Ethereum delivered by Frontier Release. It is used for running ethereum node implemented by Go. Geth provides an ethereum frontier live network. It provides a command line interface to create contracts, transfer funds, miner ether, explore block history, etc. Ethereum mist wallet is also integrated in geth.

3.2.4 Solidity

Solidity is an object oriented high level language used for implementing smart contracts for uses like crowd funding, voting, blind auctions, and multi-signature wallets [16]. It is intended to focus on the EVM. It is influenced by C++, JavaScript and python. It is statically typed, supports libraries, inheritance and complex user-defined types.

Currently, solidity is the primary language used for writing smart contracts. The contract written in solidity is compiled to EVM executable bytecode. Solidity helps developers to develop applications with self-enforcing business logic incorporated in smart contracts.

3.2.5 Opcodes in EVM

The EVM is a stack based virtual machine used for executing the smart contracts. It has a word size of 256-bits [24]. The smart contracts are converted into bytecode. When a transaction is processed, the EVM performs the calculations. Transactions may carry a payload of 0 or more bytes of data. Each opcode is encoded as one byte [24] for specifying the type of interaction with a contract and other additional information.

Others take advantage of the optimisation that can be done by fixing g as $n + 1$: in this particular setting, the g^m component may be computed as $n*m+1$, meaning one modular exponentiation and two modular multiplications instead.

Consider an example, if receiver has two messages (x and y) encrypted with sender's public key (creating cx and cy) he can add them together and send them to sender. When sender decrypts the message she will have the result of x + y. This is a simple example where the only operation is addition. Many other operations like sorting, searching, multiplication etc. can also be perform using this technique.

3.2 Implementation Tools Used

3.2.1 Truffle Framework

Truffle is a development environment, testing framework and asset pipeline for Ethereum [23]. It aims to make the life of an ethereum developer easy. It is used for compiling, linking, deploying smart contracts. The contract testing can also be automated using frameworks like mocha and chai, configure build pipeline with support for custom build processes, script deployment & migrations in framework, interactive console for direct contract communication, instant rebuilding of assets during development.

3.2.2 GoLang

Go is an open source procedural programming language, developed in 2007. It is syntactically similar to C, but has improved efficiency, safety, programming productivity, etc [12]. It provides an approach to build softwares as concurrent, garbage-collected processes. It uses packages to assemble programs, for efficient management of dependencies.

CHAPTER 3

PRELIMINARY TECHNIQUES

This chapter provides an introduction to various methodologies like paillier encryption, mining, and validation that are used throughout this thesis.

3.1 Paillier Encryption

The Paillier encryption technique [22] is a probabilistic asymmetric algorithm for public key cryptography. Key-pair based cryptography means every user has its own public and private key. The messages that are encrypted with a public key can only be decrypted with their private key.

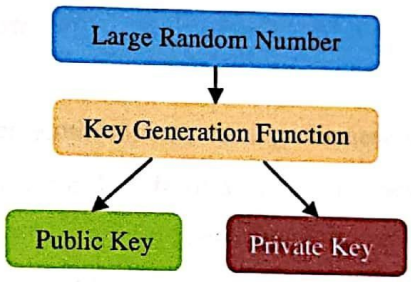


Fig. 3.1 Key-pair based cryptography

Paillier technique provides "additive homomorphism". This scheme defines a mapping **encrypt** using message to be encrypted and randomness to *cipher text*, as well as its inverse **decrypt**.

Encryption, depends on a public key consisting of a generator g and a modulus n, a pair (m, r) is mapped to c as $g^m * r^n \bmod n^2$. Some implementations compute exactly that, using two modular exponentiations and one modular multiplication.

Table 2.1: Comparison between Cryptocurrencies

CRYPTO CURRENCY	SYMBOL	HASH ALGORITHM	MEAN BLOCK TIME	MINING	TIMESTAMPING SCHEME
Bitcoin	BTC	SHA-256	10 min	Processor Intensive	PoW
Ethereum	ETH	ETHASH	14-15 sec	Memory Intensive	PoW
Litecoin	LTC	Scrypt	2.5 min	Memory Intensive	PoW
Namecoin	NMC	SHA-256	10 min	Processor Intensive	PoW
NEO	NEO	SHA-256	15 sec	Memory Intensive	PoS
Titcoin	TIT	SHA-256	1 min	Memory Intensive	PoW
Ripple	XRP	RPCA	4 sec	Not mined	

2.3 Summary

The main reason for developing a new mechanism for execution of smart contracts is to increase the throughput and efficiency. It's simply more productive to concurrently execute smart contracts. Thus, the transactions can be mined and validated simultaneously with more efficiency. Considering all the above points, this approach is taken in this thesis work.

In the next chapter, we have described the preliminary techniques that are used for implementation of the objective.

can use the presented pattern for addressing security breaches and mitigate typical attack situations.

Blumoff et al.[21] describes a multithreaded parallel programming system developed on C based runtime environment called cilk. The paper documents the productivity of of the cilk scheduler, both experimentally and analytically. It also shows that cilk computation can be used to display the execution precisely on real and engineered applications. Therefore, a Cilk programmer can concentrate on decreasing the computations and critical-path length, protected from runtime scheduling issues. The Cilk framework currently runs on a variety of work stations like the the Sun Sparcstation SMP, Connection Machine CM5 MPP, the Intel Paragon MPP and the Cilk-NOW.

In this dissertation, we developed a mechanism to improve the efficiency of the blockchain using GoLang. [12] provides the official golang implementation of the Ethereum protocol.It describes the prerequisites and detailed build instructions required for installation. It also provides a list of executables found in cmd library. It provides an overview of programming in ethereum. It is a system programming language devised by Google. It is a statically typed, compiled language similar to C, with the added benefits of garbage collection, CSP-style concurrency, memory safety, and structural typing. It uses go-routines instead of threads for control. The compiler, tools, and source code are all free and open source.

Many blockchain platforms have been developed. After studying the various research papers, Table 2.1 describes the different cryptocurrencies of various blockchains on the basis of some comparative parameters.

and accuracy. It considers those applications, including outsourced computation, where scripts require minimal time to verify.

Dickerson thomas et al. [18] proposed a mechanism for executing mining and validation mechanisms concurrently performed on the Java virtual machine and ScalaSTM. It presents a solution based on the disadvantages of serial execution like less throughput and inefficiency to attain today's synchronous multicore and congregated models. It proposes two different speculative techniques for miners and validators respectively. The miners executed smart contracts in parallel, by finding a serializable synchronous schedule for the block and allowing non-conflicting contracts to run simultanoeously. This schedule is recorded and used in validation as a fork-join program for re-implementing the miners' parallel schedule concurrently. The smart contracts were converted from solidity into scala machine to use the synchronous libraries [18]. It shows that multi-core architectures can be exploited to increase throughput of smart contract execution for both mining and validation.

Wöhrer maximilian et al.[11] used grounded theory to find the six security pattern encountered when executing the smart contracts written in Solidity. The presented pattern offers various advantages as well as drawbacks. It is used for providing solutions to some typical security issues. This method can be used by solidity developers for mitigating typical attacks. It analyzed the impact of design patterns on Ethereum platform in terms of various aspects. The design patterns described are only limited to the ethereum environment. The main issue resolved by these patterns is lack of execution control upon contract deployment, which is due to the distributed execution environment provided by Ethereum. It is a feature of Ethereum which allows programs running on the blockchain to be executed autonomously. But this has a few drawbacks too. These drawbacks appear either as harmful callbacks, adverse circumstances, or uncontrollably high financial risks at stake. The developers

for implementing complex logics. Decentralized applications are needed in areas like financial services, crowd funding, identity management, and gambling. Smart contracts are a challenging research topic that spans over areas ranging from cryptography, consensus algorithms, and programming languages to governance and law. It also summarizes the state of knowledge in this field. It provides a technical overview of Ethereum, outline open challenges, and review proposed solutions [21]. It also mentions the alternative smart contract blockchains.

Alharby maher et al.[17] conducts a systematic mapping using various techniques to study of smart contract. Smart contracts are an important feature of blockchain that is used to implement an agreement between untrusted parties without the inclusion of any third party. It identifies the current challenges in smart contracts.

Luu loi et al.[14] analyzes the degree to which cryptocurrency framework can uphold the right semantics of scripts. It shows that there are more chances of occurrence of an attack when the execution of script requires non trivial computation. In case of attacks, the miners' computational resources gets wasted and as a result incorrect script results are also accepted. These attacks drive miners to a doomed decision, which is called the verier's dilemma, whereby rational miners are well-incentivized to accept unvalidated blockchains. The framework of computation through a scriptable cryptocurrency is called a consensus computer and model is built that captures incentives for verifying computation in it. It proposes a solution to the verifier's dilemma which incentivizes correct execution of certain applications, including outsourced computation, where scripts require minimal time to verify. It also discussed two distinct, practical implementations of our consensus computer in real cryptocurrency networks like Ethereum. The correctness of computations performed on a consensus computer are affected by incentive structure and attacks .The ε-consensus computer is implemented in Ethereum with various trade-offs in latency

It described the typographical conventions for the formal notation. It explained the procedure of the execution of a transaction which is the most typical part of the Ethereum protocol. It also uses formal semantics to explain the contract creation process with the help of intrinsic parameters. The paper also describes basics of the execution model of ethereum. It also explains Proof-of-Work which is a cryptographically secure nonce that demonstrates the amount of calculation that has been exhausted in the assurance of some token esteem n. It is used to authorize the blockchain by giving significance and assurance to the notion of difficulty. It also explains the key features of ethereum project and how it differs from the other cryptocurrencies. The paper provides a list of terminologies of the project with description.

Hildenbrandt everett et al. [19] discovered the capabilities of the widespread practical adoption Ethereum. It also analysed the ascent in the security vulnerabilities and high prone contract failures and their impact on Ethereum. In this work, KEVM is presented. It is an practicable formal semantics of the EVM. This semantics were created in a framework for executable semantics, called the K framework. It asserted and validated the computational feasibility of their approach, by testing KEVM's generated EVM interpreter. It was demonstrated that the semantics is workable against a various test cases on various machines. The authors also demonstrated the KEVMs' extensibility using software analysis tools by monitoring the relevant properties of interest that computes gas during execution. It helps reveal obscurities and potential sources of error in the existing paper used for formalizing the EVM semantics. It describes the features of KEVM that make it an ideal formal reference.

Tikhomirov sergei [20] described ethereum as an efficient blockchain platform for smart contracts. It maintains a P2P network to record a common view of the global state and executes code upon request. The states are stored in a blockchain secured by a PoW consensus scheme. Ethereum is a full-featured programming language used

CHAPTER 2
LITERATURE REVIEW

2.1 Overview

Various research papers, review papers have been studied in order to study various blockchain platforms. It also provides a summary of ethereum blockchain, EVM and also discusses the execution mechanism of opcodes being invoked in EVM.

2.2 Related Literature

Cryptocurrencies are encrypted, peer-to-peer networks which are becoming a fast and comfortable means of payment, with a worldwide scope. They are not granted by any central authority, hence they are free from all governmental supervision. The cryptocurrency coins are validated by blockchain. Blockchain is an incorruptible digital ledger technology. Many blockchain platforms have been developed each trying to be better than the other.

In this section we discuss about ethereum and EVM and work done to improve the mechanism for execution of opcodes in EVM. Lack of concurrency in EVM architecture is one of the major concerns of the ethereum blockchain, as it limits throughput. Research is being done to propose new mechanism to improve the smart contract execution to make it more efficient and fast.

Wood [6] described the Ethereum blockchain paradigm in detail. He discussed the factors responsible for the development of ethereum like geographical separation, interfacing docility, or may be the incompatibility, inadequacy, reluctance, cost, corruption of existing systems, vulnerability, or inconvenience. It provided an overview of the basic concepts behind Ethereum, the transaction-based state machine.

After a block time of 14-15 sec, the miner nodes in the P2P network try to transform the state of associated contracts in the block. If a miner is successful in solving the complex computational problem, that miner broadcasts the block to all the nodes with a happen-before graph. The validators use this graph to implement the blockchain again. This is done by re-executing the miner's parallel schedule concurrently. The nodes accept the block and add it to their record. By adding parallelism to smart contract execution, better response can be achieved in less time.

1.9 Thesis Organization

This thesis is organized in 6 chapters: Chapter 1 describes an overview of blockchain, ethereum and its components followed by the problem statement and the research objectives. Chapter 2 provides a description of the related work done in this field. Chapter 3 and 4 explain the various steps which make up the proposed mechanism. Chapter 5 presents the simulation results and analysis for the validity of the new model, followed by conclusion and future aspects. The codes, outputs and calculations are contained within the Appendices.

1.10 Summary

In this chapter, an introduction of blockchain, ethereum and its execution components have been discussed. The general working of smart contract and ethereum are also described to clarify the objectives.

In the next chapter, based on the motivation described above we have summarized the various research papers, review papers used to study ethereum.

virtual machine embedded within Ethereum node for executing contract bytecode. Smart contracts are the programs that run on top of the blockchain. The aim is to facilitate, execute and authorize an understanding between untrusted parties, however without the inclusion of any third party. They implement an association with cryptographic code. These smart contracts are executed sequentially. This whole execution process incurs more time and provides less efficiency. The research is to look for an efficient mechanism so that the performance is not overlooked.

1.7 Research Objectives

The objective of this dissertation is to improve the mechanism for execution of opcodes in EVM. The broad objective is to study about Ethereum. The specific objectives of the study were:

1. To analyze the working of Ethereum cryptocurrency.
2. To execute opcodes in EVM using Go-lang.
3. To propose a new mechanism of opcodes in EVM for better execution.
4. To propose a new mechanism for adding concurrency to smart contracts.

1.8 Methodology

In the dissertation, we proposed a scheme which focused on the improvement in the execution speed of the miners and validators in the P2P network of the ethereum blockchain. A transaction is initiated between two parties using ethereum mist wallet. The address of the sender's wallet is linked with the private key. The transaction is then signed by the sender using his digital signature and is processed in the EVM. The EVM processes the transactions into opcodes. The opcodes are executed parallelly using go-routines. The nodes accumulate the new transactions into a block.

every transaction. It is assumed that a group of the transactions is authentic, if there are no dependencies among them. Thus, sequential execution of these transactions is correct. But, this execution mechanism limits the throughput and efficiency of the blocks. As a result, the efficiency of whole blockchain is affected.

1.5 Problem Identification

In today's digital world, the secured exchanges between untrusted parties are executed through scripts called smart contracts by blockchain platforms like Ethereum. In the P2P network, the smart contracts enable miners and validators to communicate with one another. The execution of smart contracts occur multiple times but without concurrency. The miners execute these contracts serially [17] before adding them to the blockchain. Afterwards, the validators re-execute those contracts for checking whether these contracts have been executed correctly by miners or not. It fails to exploits today's synchronous multicore and cluster model [18]. Hence, the problem of our research is:

- The executions of smart contracts in ethereum are less efficient.
- EVM takes more time to execute the smart contracts.
- The current execution mechanism used in mining and validation poses many security issues.

1.6 Motivation

In the current digital scenario, mechanisms are needed that are more secure, efficient and take less execution time. Blockchain platforms have gained immense popularity in the recent years due to their heightened privacy features and decentralized nature. For instance, ethereum is the world's first blockchain used for secure exchange of information which is driven by currency called ether. The EVM is a stack-based

miner is successful in solving the complex computational problem, that miner broadcasts the block to all the nodes. The nodes accept the block and add it to the ledger making the transaction complete.

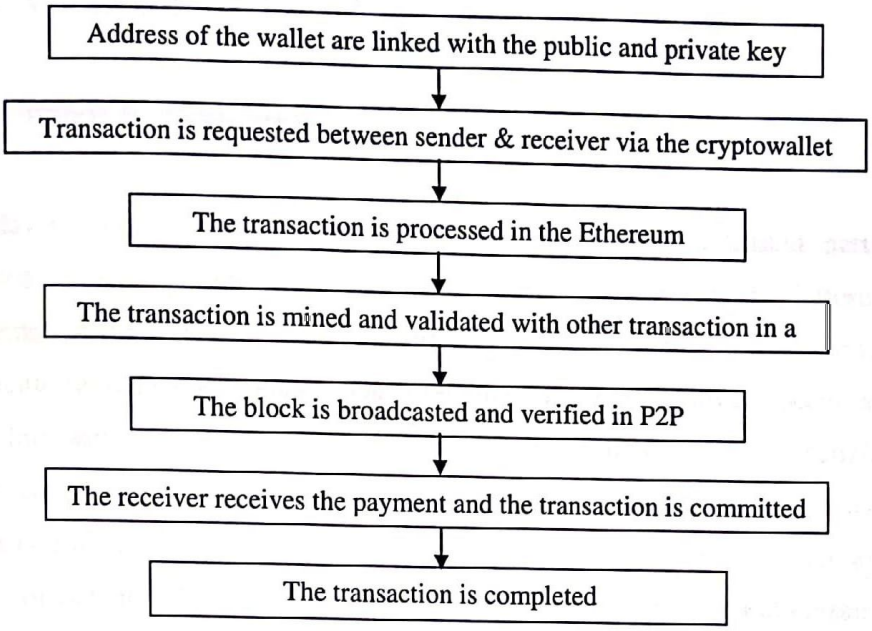


Fig. 1.5: Work flow diagram of Ethereum

The P2P nodes are responsible for transferring the shift from state to state, rather than any central authority. The smart contract are executed according to the rules initially programmed by the developer. Actual computation of smart contracts on the EVM is obtained through bytecode, but the smart contracts are written in high-level languages such as serpent and solidity [16].

1.4 Challenges in execution of Smart Contracts

The execution of smart contracts in EVM depends on the language of the EVM. Most of the languages in which EVM are written are serially execute where each transaction is executed consecutively without any interference from other transactions. In sequential execution there is complete isloation between each and

are not produced in the same manner as the regular money. No central authority is involved in issuing new currency, hence it is free from governmental supervision.

Each action that modifies the ethereum's state costs some ether as a fee. The miners who are able to solve the computations and append a block in the chain are rewarded ether as account of transaction fees. Table 1.1 describes the various sub-denominations of ether cryptocurrency. Ether can easily be converted to dollars or other traditional currencies through Crypto-exchanges [16]. One Ether costs approximately $193.21 USD and ₹13,601 INR. Ethereum has a metric system which is used as units of ether. The smallest value of ether is called Wei. One Ether equals 10^{18} Wei.

Table 1.1: Subdenominations of Ether [6]

MULTIPLIER	NAME
10^{10}	Wei
10^{12}	Szabo
10^{15}	Finney
10^{18}	Ether

1.3.4 Working of Ethereum

With ethereum, every time a program is used, a network of thousands of computers processes it. In the figure 1.5, the general working of ethereum is described. A transaction is initiated between two parties using a cryptocurrency wallet. The address of the sender's wallet is linked with the private key. The transaction is then signed by the sender using his digital signature and is processed in the EVM. The signed transaction is broadcasted in the peer-to-peer network. The nodes collect the latest transactions in a block. After a block time of 14-15 sec, the miner nodes in the P2P network try to process the block using a specific time-stamping scheme. If a

transaction is submitted, the transaction is not executed immediately instead is it pooled in a transaction pool. These transactions are not yet executed and not yet written to the Ethereum ledger. EVM nodes are similar to mining nodes however they do not do mining. They are primarily responsible to provide a runtime that can execute code written in smart contracts. It can access accounts contract and externally owned its own storage data. It does not have access to ledger but has limited information about current transaction.

In the figure 1.4, the execution process of ethereum is described with respect to EVM. Contracts written in a smart contract-specific programming language are compiled into 'bytecode', which the EVM can read and execute. All the nodes execute this contract using their EVMs. Every node in the network holds a copy of the transaction and smart contract history of the network, in addition to keeping track of the current 'state'. Every time a user performs some action, all of the nodes on the network need to come to agreement that this change took place.

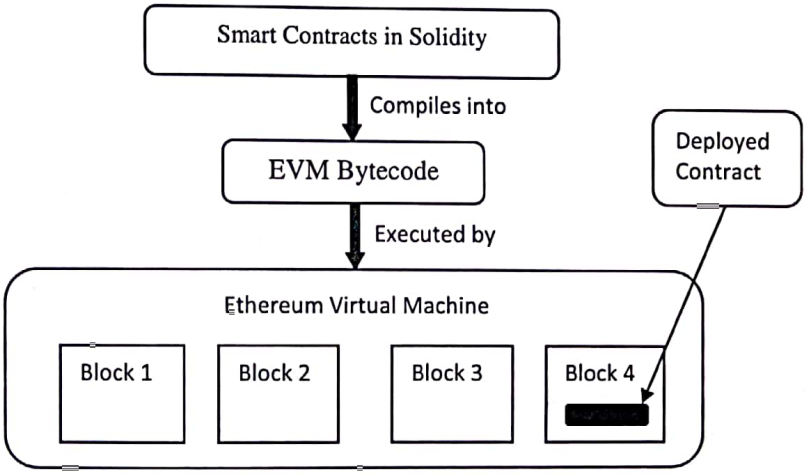


Fig. 1.4: Execution process of Ethereum

1.3.3 Ether

Ether is a fuel used for performing operations on Ethereum. It is a form of payment used by the clients for executing the requested operations on the machine [16]. They

javascript, Ruby, rust, etc. The Figure 1.3 provides the example of a smart contract written in solidity on go-ethereum (GETH) [13]. In this figure 1.3, a contract "AddInteger" is executed upon initialization which takes two integers as input and returns the sum as output.

```
contract AddInteger{
uint private c;
function addition(uint _a, uint _b) public constant
returns(uint)
{
    c = _a+_b;
    return c;
  }
}
```

Fig. 1.3: Example of Smart Contract

As long as the network exists, the smart contracts are executable, and will get deleted only when they are programmed to be destroyed [14]. The smart contracts are executed first by miner nodes in the P2P network. They try to process the block using a specific time-stamping scheme. If a miner is successful in solving the complex computational problem, that miner broadcasts the block to all the nodes and is rewarded some ether in return. The broadcasted block is then added to the blockchain. The validator then, processes the full blockchain and checks the data structure and predicates of each block. The efficiency of smart contract execution leads to the efficiency of the overall blockchain.

1.3.2 EVM

EVM is a virtual machine that serves as the execution component in Ethereum. It plays an important role in construction of ethereum, because it is responsible for handling internal state and computation on the network. It also handles the account information like addresses, balances, current gas price, and block information. The purpose of EVM is to execute the smart contracts line by line. However, when a

Ethereum is a stack of layers built on top of each other. Figure 1.2 describes the architectural stack of ethereum blockchain. The first layer makes everything possible. It comprises of an extensive computer network that processes transactions and keep a shared database updated over time. The second is the software layer that allows developers to run programs called smart contracts on the Ethereum blockchain, using a programming language called Solidity. The last layer comprises of applications offering services to ethereum users. This remarkable architecture lacks a central point of failure and is somehow "unstoppable".

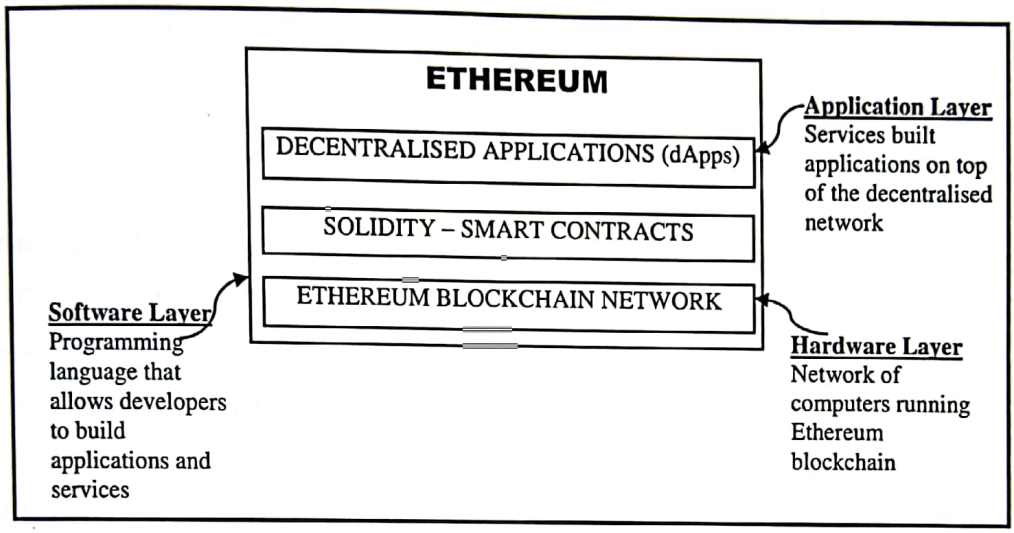


Fig 1.2: Ethereum architectural stack

1.3.1 Smart Contracts

Smart Contracts [11] act as an additional layer in the basic architectural stack of a blockchain system. They are account holding objects on the ethereum blockchain. They can interact with different contracts, decide, store information, and send ether to others. Contracts are defined by their creators, however their execution and the extension services are provided by the ethereum network itself. Ethereum is implemented in various programming languages like: Golang [12], C++, python,

1.3 Ethereum

Most of the online businesses, services, etc. are based on a centralized system of governance. This approach has been used for hundreds of years, and while history proved time and time again that it's flawed, its implementation is still necessary when the parties don't trust each other. To solve the security breaches, a decentralized system should be implemented. For example, Blockchain is a secured platform for transactions which has replaced the central administration with new tools for improved authentication and authorization, therefore formalizing and securing new digital relationship. Earlier Blockchain technology was primarily used as a substitute of regular money and therefore was referred as a medium of payment transaction. But now numerous new blockchain platforms have been created which are beyond just supporting a digital currency. For instance, ethereum blockchain is used for secure exchange of information.

Ethereum blockchain [6] is an open source platform. It executes smart contracts. A smart contract [11] is a computer protocol used to digitally facilitate, verify, or implement the negotiation or execution of a contract. The valid transactions are executed without the participation of any third party. The main objective of smart contracts is to provide better security than the traditional contract laws and reduce the transaction costs associated with contracting. In the P2P network, the smart contracts enable miners and validators to communicate with each other.

The execution of smart contracts is done in Ethereum Virtual Machine (EVM) [6] which is a virtual execution component in ethereum. Further, the execution of these contracts depends on the language of the EVM. Most of the languages in which EVM are written are serially executed where each transaction is carried out continuously with no impedance from other transactions.

Figure 1.1 describes the workings of the centralized and decentralized ledger systems. In a centralized system the participants of the network share the same ledger copy. While in the decentralized system, every node of the network can access the data shared across that network and can possess an identical copy of it. Further, any modifications made to the record are reflected and replicated in the ledger copies of all participants within seconds or minutes. Once the information is stored, it becomes permanent. Thus, it becomes more difficult to attack the distributed unlike the centralized ledgers because all the distributed copies need to be attacked simultaneously for an attack to be effective. Further, these records are resistant to harmful changes by a single party.

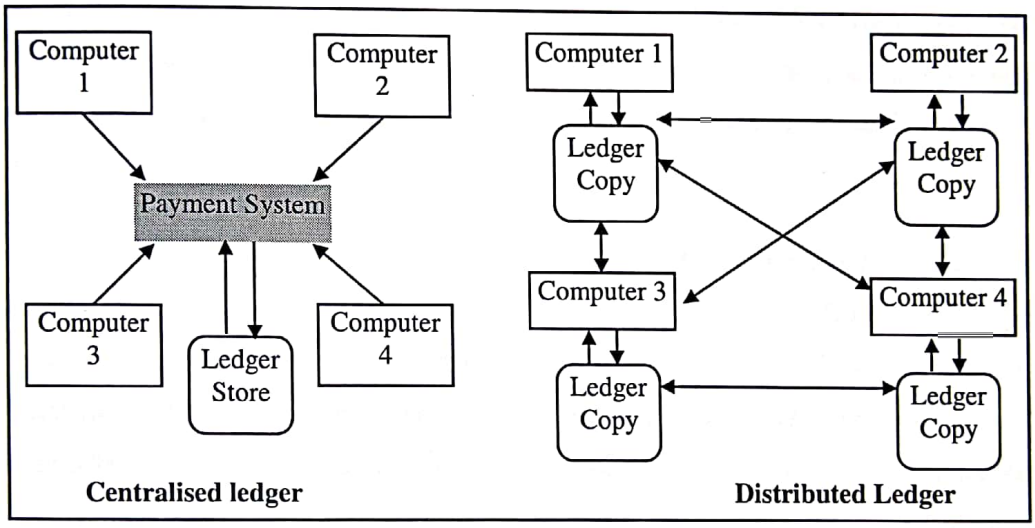


Fig. 1.1: Centralised Ledger and Distributed Ledger mechanism

Earlier Blockchain technology was primarily used as a substitute to regular money and therefore was referred as a medium of payment transaction and store of value. But, nowadays many new blockchain platforms have been created which are beyond just supporting a digital currency. For instance, ethereum blockchain is used for secure exchange of information. Ethereum is a platform used for facilitating peer-to peer contracts (P2P) [10] and applications by means of cryptocurrency called ether.

Blockchain's evolution can be categorized into three eras namely cryptocurrencies, contracts, and distributed applications. The blockchain technology was first, used in the development of cryptocurrencies. Cryptocurrencies enables people to make transactions avoiding standard currencies. They are digital money created from the execution of code during online transactions [4] done in fields like blockchain etc. They have inbuilt cryptographic tools which are used for securing and verifying payments as well as for controlling the creation of new units of a particular cryptocurrency.

This technology was accepted a few years ago and facilitates digital barter system. It has allowed valid transactions between parties without the need of a third party. This decentralized automation appeared with Bitcoin [5], the first and the most prevalent cryptocurrency. Its immense success has led to the emergence of thousands of new cryptocurrencies in the global market like ethereum [6], altcoin, namecoin [7], litecoin [8], etc. Each cryptocurrency coin is validated by an incorruptible decentralized digital ledger technology.

1.2.1 Centralized and Decentralized Ledger Technology

The traditional method of transactions was based on the centralized ledger systems [9]. In this framework, a single database is used for maintaining the records of transactions which is supervised by a central authority. While on the other hand, in a decentralized framework [9], there is a distributed ledger whose copy is consensually shared and synchronized across network spread over multiple sites and geographies. It maintains the transactions in a distributed manner thus, eliminating the need for keeping a check against manipulation. They are not issued by any central administration, hence they are free from all governmental supervision.

CHAPTER 1
INTRODUCTION

1.1 Overview

Internet has been one of the greatest discoveries of the past decades. Its remarkable advancement has created immense opportunities for developing new prospects of information across the globe. It has created new ways for people to communicate, gather and share data of their social life. But due to the rapid computer advances there has been a surge in the number of endpoints and potential ways for cybercriminals to access to private information. This has led to the requirement for a more secured technology to protect information from the cybercriminals. To overcome, these security breaches and attacks various security mechanism have been proposed [1]. Still as the technology is progressing the breaches are also increasing. The blockchain technology provides secure and robust mechanism to overcome these problems.

In this chapter a background introduction of blockchain and ethereum is explained along with the description of current execution mechanism of opcodes and smart contracts in EVM. It finally outlines the challenges, problem statement, objectives of the research, followed by the methodology and motivation.

1.2 Blockchain

Blockchain [2] is a secured platform for transactions which has replaced the central administration with new tools for authentication and authorization, therefore formalizing and securing new digital relationship [3]. It has the potential to fix the damage that the modern technologies and science have caused to the world.

CONTENTS

List of Tables

List of Figures

List of Abbreviations

Acronym	Description
EVM	Ethereum Virtual Machine
SHA	Secure Hash Algorithm
RPCA	Ripple Protocol Consensus Algorithm
ETH	Ethereum
PoW	Proof-of-Work
PoS	Proof-of-Stake
P2P	Peer-to-Peer
GETH	Go-Ethereum
JVM	Java Virtual Machine

ABSTRACT

In the current digital world, the requirements of security and authenticity of the users is inevitable, for which many new technologies have been developed like cryptographic tools, blockchain, etc. The most recent one is blockchain, which is an incorruptible digital ledger used for storing financial transactions as well as private information. Smart contracts are the programs that run on top of the blockchain. The aim is to facilitate, execute and authorize an understanding between untrusted parties, however without the inclusion of any third party. They implement an association with cryptographic code. These smart contracts are executed without real concurrency. Before appending the smart contracts to the blockchain, they are first sequentially executed by miners. Afterwards, those contracts are re-executed sequentially by validators for checking whether these contracts have been executed correctly by miners or not. This whole process incurs more time and provides less efficiency. So, in this dissertation, we have proposed a new mechanism which provides better security and throughput in lesser time.

ABSTRACT

In the current digital world, the requirements of security and authenticity of the users is inevitable, for which many new technologies have been developed like cryptographic tools, blockchain, etc. The most recent one is blockchain, which is an incorruptible digital ledger used for storing financial transactions as well as private information. Smart contracts are the programs that run on top of the blockchain. The aim is to facilitate, execute and authorize an understanding between untrusted parties, however without the inclusion of any third party. They implement an association with cryptographic code. These smart contracts are executed without real concurrency. Before appending the smart contracts to the blockchain, they are first sequentially executed by miners. Afterwards, those contracts are re-executed sequentially by validators for checking whether these contracts have been executed correctly by miners or not. This whole process incurs more time and provides less efficiency. So, in this dissertation, we have proposed a new mechanism which provides better security and throughput in lesser time.

ACKNOWLEDGEMENTS

Throughout my dissertation, I have been very fortunate to get help and support from many people. I feel immense pleasure in expressing my profound sense of gratitude to my supervisor Mrs. Aarti Gautam Dinker, SOICT for her guidelines, useful suggestions and for permitting me to carry out this work which helped me in completing the work on time. She inspired me greatly to work on this dissertation. Her willingness to motivate me contributed tremendously to the work. I would also like to thank her for showing me few examples that are related to the topic in my dissertation.

I would like to acknowledge Dr. Anurag Baghel, HOD of CSE Department for his encouragement that boosted my morale and confidence. I would like to express my gratitude and heartfelt thanks to Dr. Indu Uprety, Dean of ICT for her excellent supervision and support in this research.

Besides, I would like to thank the authority of School of Information and Communication Technology (ICT), Gautam Buddha University for providing a good environment and facilities that helped me complete this dissertation.

I would like to express my heartfelt thanks to my beloved parents for their blessings and wishes for the successful completion of this work. Last but not the least, Almighty God has always been the invisible divine force behind my accomplishments.

Vartika Agarwal

14/ICS/068

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

GAUTAM BUDDHA UNIVERSITY

GAUTAM BUDDHA NAGAR-201312

Certificate

It is certified that the work done entitled "EFFICIENT SMART CONTRACTS EXECUTION IN ETHEREUM USING CONCURRENT MULTICORE ARCHITECTURE" by "Vartika Agarwal" Roll No. 14/ICS/068 Integrated M.Tech final year with specialisation "SOFTWARE ENGINEERING" has been carried under my supervision.

Mrs. Aarti Gautam Dinker

Faculty Associate

School of ICT

Gautam Buddha University

Greater Noida, 201312

(U.P) INDIA

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

GAUTAM BUDDHA UNIVERSITY

GAUTAM BUDDHA NAGAR-201312

Candidate's Declaration

I hereby declare that the work embodied in this dissertation entitled "EFFICIENT SMART CONTRACTS EXECUTION IN ETHEREUM USING CONCURRENT MULTICORE ARCHITECTURE" is submitted for the partial fulfillment of requirements to award the degree of Integrated M.Tech (Software Engineering) to School of Information and Communication Technology (ICT), Gautam Buddha University, Greater Noida. It is an authentic record and my own bonafide work has been carried out under the supervision of Mrs. Aarti Gautam Dinker, School of ICT. This work is correct to the best of my knowledge and belief and has been undertaken taking care of engineering ethics. It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of any university or other institute of higher learning, except where due acknowledgement has been made in the text. Responsibility for any plagiarism related issues stands solely with me.

Name of Student : Vartika Agarwal

Roll No : 14/ICS/068

Signature : Vartika

Date : 23/05/19

Place : Greater Noida

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

GAUTAM BUDDHA UNIVERSITY

GAUTAM BUDDHA NAGAR-201312

Candidate's Declaration

I hereby declare that the work embodied in this dissertation entitled "EFFICIENT SMART CONTRACTS EXECUTION IN ETHEREUM USING CONCURRENT MULTICORE ARCHITECTURE" is submitted for the partial fulfillment of requirements to award the degree of Integrated M.Tech (Software Engineering) to School of Information and Communication Technology (ICT), Gautam Buddha University, Greater Noida. It is an authentic record and my own bonafide work has been carried out under the supervision of Mrs. Aarti Gautam Dinker, School of ICT. This work is correct to the best of my knowledge and belief and has been undertaken taking care of engineering ethics. It contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of any university or other institute of higher learning, except where due acknowledgement has been made in the text. Responsibility for any plagiarism related issues stands solely with me.

Name of Student : Vartika Agarwal

Roll No : 14/ICS/068

Signature : Vartika

Date : 23/05/19

Place : Greater Noida

EFFICIENT SMART CONTRACTS EXECUTION IN ETHEREUM USING CONCURRENT MULTICORE ARCHITECTURE

A dissertation report submitted in partial fulfillment of the requirement for the award of the degree of

MASTER OF TECHNOLOGY

in

SOFTWARE ENGINEERING

SUBMITTED BY:

VARTIKA AGARWAL
14/ICS/068

SUPERVISED BY:

MRS. AARTI GAUTAM DINKER

(Faculty Associate)

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

GAUTAM BUDDHA UNIVERSITY

GREATER NOIDA - 201312, UTTAR PRADESH, INDIA

MAY, 2019

EFFICIENT SMART CONTRACTS EXECUTION IN ETHEREUM USING CONCURRENT MULTICORE ARCHITECTURE

A Dissertation report submitted in partial fulfillment of the requirements for the award of the degree of

Master of Technology

in

SOFTWARE ENGINEERING

Submitted By:

VARTIKA AGARWAL

14/ICS/068

Supervised By:

MRS. AARTI GAUTAM DINKER

(Faculty Associate)

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

GAUTAM BUDDHA UNIVERSITY

GREATER NOIDA-201312, UTTAR PRADESH, INDIA

MAY, 2019