MapReduce-Based Framework For Blockchain Scalability

Maneesh Darisi, Om Modi, Vasu Mistry, and Dhiren Patel

Abstract The proliferation of blockchain technology into the wide spectrum of industries has been stymied by its inability to scale. With the augmenting popularity of blockchains, scalability of blockchain is hindering it from attaining the meteoric transaction rate present in the existing solutions like MasterCard and VISA. Presently, the scalability solutions of blockchains use several off-chain and on-chain mechanisms. This paper proposes an on-chain solution backed by big data technologies. We aim to provide a real-time scalable transaction processing by using big data framework to overcome the roadblocks of scalability. Our framework has horizontal and vertical scaling to augment the sluggish blockchain transaction rate using *sharding* frameworks of big data.

Keywords Blockchain scalability · Big data · Sharding · MapReduce

1 Introduction

Blockchain is a sequential agglomeration of an immutable data structure called blocks that includes the set of valid transactions which is transparent to every member of the chain and has entities like miners who perform consensus and deploy blocks on the chain facilitating the security from single point of failure.

M. Darisi · O. Modi (✉) · D. Patel
Department of Computer Engineering and Information Technology, Veermata Jijabai Technological Institute, Mumbai 400019, India
e-mail: om.vjti@gmail.com

V. Mistry
Independent Researcher, Mumbai, India

D. Patel et al. (eds.), *IC-BCT 2019*, Blockchain Technologies,
https://doi.org/10.1007/978-981-15-4542-9_11

A blockchain is a tuple (G, B) where G is a genesis state and $B = [\beta 1, \beta 2, \beta 3 \ldots]$ is an ordered list of blocks. A blockchain is valid if every $\beta \in B$ is valid, and so $G + \beta 0 + \beta 1 + \cdots = \sigma f$ is a valid state. A block β is a package containing a list of transactions T, a reference to a parent block and auxiliary verification data. A block is valid in the context of a state σ if:

• The block's transaction list is valid in the context of σ

• Some other conditions, generally determined by the consensus algorithm (e.g. proof of work), are met. Buterin [1]

It has been a decade ever since blockchain has come into this technology world but has still failed to throw light on some burning issues like scalability, interoperability, governance, etc. Scalability in the context of blockchains is dependent on security and capacity limitation of public blockchains protocol which demands:

- Every transaction on the blockchain must be processed by every single node on the blockchain.
- Every operation like payment and deployment of a smart contract must be replicated by all the full nodes.

This makes the public blockchain autonomous and reliable and eliminates the need for dependence on any counterparty. But the above protocol depreciates the throughput of the blockchain drastically. The proof–of-work (PoW) [2] protocol puts the constraint onto the blockchain that the blockchain throughput is equal to the processing capability of the individual nodes only. We require blockchains to scale to the global user database and at the same time ensure as much decentralization as possible.

Presently, the scalability solutions of blockchain are divided into two main categories:

1. On-chain solutions: modifying the underlying blockchain infrastructure.
2. Off-chain solutions: develop additional infrastructure that connects to the blockchain.

Ethereum [3] projects relating to on-chain solutions focus on the changing of the present consensus algorithm from proof of work [2] and to proof of stake [4] (PoS) and implement sharding on to the blockchain infrastructure. Ethereum projects relating to off-chain solutions which focus on implementing state channels, lightning network called Raiden [5], a new concept layer to scale smart contracts called plasma [6], and Truebit [7] to process complex computations off the chain. Scalability of blockchain systems is heavily affected by three factors—decentralization, scalability and security.

Due to its operating nature, colossal amount of data is generated, and to deal with it, big data provides the tools which are highly scalable and powerful for executing distributed parallel computations on the data.

MapReduce [8] is a stalwart and robust framework that provides a distributed and parallel environment for scalable and fault-tolerant computations of batch jobs. In the mapper phase, the data is divided into shards and has a <key, value> format which is given to reducer for its aggregation and giving the overall output of the job. Scalability is ensured as the mappers and reducers execute across the cluster parallelly in a distributed manner. The concurrency and task allocations are performed intrinsically and securely by MapReduce.

Hadoop [9] provides inherently the benefits which are required for the blockchain to scale. Hadoop works best in a cluster by using low-cost commodity hardware and is buttressed by the parallelism provided by MapReduce. Hadoop at its core is majorly made up of HDFS and MapReduce. HDFS filesystem provides fault tolerance and security which is required while computation of transactions taking place in the Hadoop ecosystem. HDFS by itself supports several cryptomechanisms to secure data using encryptions and authentication frameworks. Thus, by utilizing low-cost commodity hardware and being able to run these scalable Hadoop ecosystems, decentralization can be achieved, while scalability of transactions is possible by using the MapReduce framework. Knox [10] and Ranger [11] intrinsically provide security when parallel computations take place in the Hadoop environment. We aim to break the trilemma by using ecosystem to counter the scalability problem of the blockchain.

We aim to integrate big data technologies on to the blockchain architecture to improve the present blockchain transaction rate. We propose the architectural and technological changes required in the network and consensus layer to scale the present blockchain architecture.

Rest of the paper is organized as follows: Sect. 2 discusses related work in the field of blockchain scalability, and Sect. 3 describes the proposed solution with conclusion and references at the end.

2 Related Work

Practically, it has been proven that proof of work is sluggish time and resource consuming. Therefore, a new wave of consensus algorithms aims to overthrow proof of work. Delegated proof of stake, Byzantine fault-tolerant [12] variants, etc., aim to obtain only the consensus of only a representative group of entities rather than the entire network. These systems abate the decentralization aspect of blockchain in order to increase scalability.

Polkadot [13] and Cosmos [14] aim to build a network infrastructure which to resolves interoperability issues among heterogeneous blockchains. Polkadot has developed a relay chain and several parachains for provisioning interoperability and scalability, but Polkadot is not envisaged to support the deployment of customized complex blockchains smart contracts. Cosmos is an Internet of blockchains scalability network that achieves scalability by using high performance, secure tendermint [15] consensus core which provides a transaction rate up to 10000 tx per second. Cosmos implements a intrinsic star topology to connect heterogeneous independent blockchains(zones) using a central hub. Polkadot and Cosmos are not envisaged to support the deployment of customized complex blockchains smart contracts.

Zilliqa [16] and Merklix tree [17] envision to implement sharding onto the existing blockchain platforms. The Merklix tree aims to increase the torpid transaction of bitcoin using sharding. Zilliqa and Merklix tree compromise on security aspect of the blockchain.

Cardano [18] is an innovative project which not only addresses scalability and interoperability issues of blockchain but also aspires to improve the governance of the existing blockchain system.

3 Proposed Solution

Our solution solves one of the most difficult conundrums hindering the proliferation of blockchain technology in the industry by using big data. We address the problem of scalability while not compromising on decentralization using the existing Hadoop scalable tools. Our solution implements transaction and network sharding to ameliorate the torpid blockchain rate.

Since we are using *MapReduce,* all the inputs to *MapReduce* framework should be in a <key, value> pair format.

According to Fig. 1, the *MapReduce* workflow consists of the:

- Mapper phase
- Reducer phase
- Final Consensus phase.

There are two input pools to our framework which are

1. Transaction pool
2. Miner pool.

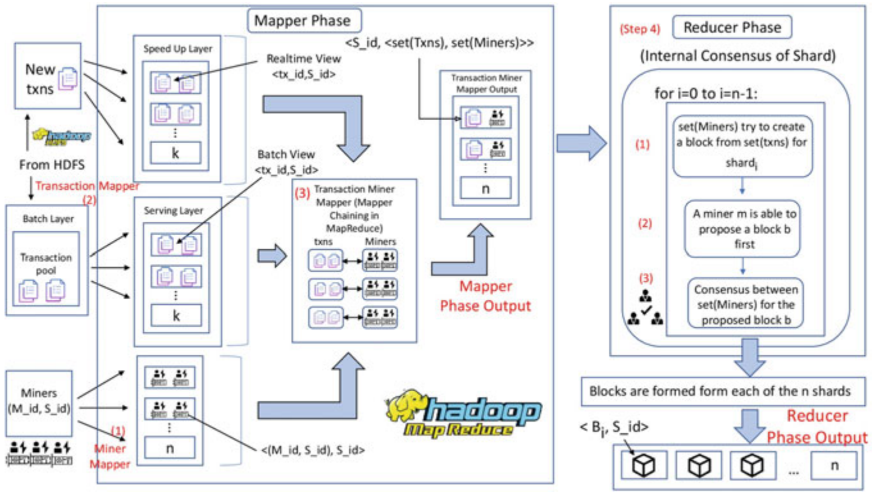
Mapper Phase
<S_id, <set(Txns), set(Miners)>>
New txns
From HDFS
Transaction Mapper (2)
Batch Layer
Transaction pool
Speed Up Layer
k
Realtime View <tx_id,S_id>
Batch View <tx_id,S_id>
Serving Layer
k
(3) Transaction Miner Mapper (Mapper Chaining in MapReduce)
txns
Miners
Transaction Miner Mapper Output
n
Mapper Phase Output
Miners (M_id, S_id)
(1) Miner Mapper
n
<(M_id, S_id), S_id>
hadoop Map Reduce
(Step 4) Reducer Phase
(Internal Consensus of Shard)
for i=0 to i=n-1:
(1) set(Miners) try to create a block from set(txns) for shard$_i$
(2) A miner m is able to propose a block b first
(3) Consensus between set(Miners) for the proposed block b
Blocks are formed form each of the n shards
< B$_i$, S_id>
Reducer Phase Output
...
n

Fig. 1 Detailed implementation workflow architecture of the mapper and the reducer phase

Step 1: To bifurcate the miner pool into ‘n’ shards (according to Eq. 1) using random sampling rather than applying geographic sharding. The major disadvantage of geographic sharding is the fact that the miners bifurcated based on their geographic locations reducing the decentralization of blockchain systems. If the shards are created based on location, then the mining pool of a certain location can have the authority over the internal consensus of that shard. It is better to randomly sample the miners based on their quantity into n shard, thus avoiding the concentration of authority and increasing the security of the system. But this leads to the issue network propagation delay among the miners in the shard. All the new miners added to the mining pool are taken to pre-computed shards after each epoch.

Step 2: The transaction pool is divided into ‘k’ shards (according to Eq. 1). Two types of sharding can be applied on the transaction pool 1. state-based sharding and 2. transaction-based sharding. Our framework is presently using transaction-based sharding and the lambda architecture to dynamically update the transaction pools [19, 20]. The batch layer creates batch views of the transactions that have entered the transaction pool and sends it to the service layer for further processing. Speedup layer produces real-time views and sends them to the transaction-miner mapper directly along with the batch views, thus reducing latency time. All the batch views are updated after each epoch.

Step 3: The transaction-miner mapper maps the k%n transaction shards to the n miner shards.

Step 4: In the reducer phase, the miners in each miner shard must come to consensus upon the k%n transaction shards and have to propose a block. At the end of the reducer phase, each reducer would have proposed a block which had received the consensus from the miners present in its shard.

Step 5: In the final consensus phase, we aim to use Byzantine fault-tolerance algorithms/proof of stake algorithms in order to obtain the consensus of all the shards on the blocks proposed by the reducer phase.

The consensus algorithm we apply in the final consensus phase and reducer has an heavy impact on the performance and throughput transaction rate which we obtain at the end All the intershard and intrashard communication are handled by Apache Kafka [21] using the publisher subscriber models.

3.1 Actors of Solution

In the rest of the solution, we will use these actors:

- Miner
  Miners validate new transactions and propose new blocks and record them on the global ledger and receive an incentive when they solve the complex mathematical problem.

- Leader
  Leader chooses ‘h’ members from each shard to form a validation group.
- Validator
  These are miners are subset of global miner list who validate blocks proposed by each shard which are added to the global ledger.

3.2 Updation of Transaction Pool

Lambda architecture can be used for efficient and real-time batch processing of transaction pool. It mainly consists of these three layers: batch layer, service layer, speedup layer.

Speedup Layer
This layer is primarily used to provide low latency updates. Speedup layer uses an incremental computation approach rather than a batch computation approach. Speedup layer relies on the fact that the transaction data stored is transient and modicum in nature. Processing data on a smaller scale provides greater design flexibility. Speedup allows complex computations to take place on these real-time views. Speedup layer aims to provide low latency updates to the batch views. Therefore, all the real-time views are directly supplied to the transaction-miner mapper.

There two major functions of the speedup layer:

- Storing the real-time views.
- Processing the input data stream so as to update the real-time transaction views.

Batch Layer
The batch layer is used to store the immutable growing transactional pool. The batch layer is responsible for creating real-time batch views for the service layer to process the blockchain transactions. The batch layer needs to be able to do two things:

- Store an immutable, constantly growing transactional pool.
- Create and continuously update batch views for the service layer.

The serial processing of transactions results in low scalability; thus, we aim to do batch processing and process these blockchain transfers using MapReduce. MapReduce is best done using batch processing paradigm.

Service Layer
Service layer is used to process the batch views that it receives from the service and speed layer. When new batch views are available, the serving layer automatically swaps in so that more up-to-date results are published on to the blockchain. The service layer is distributed among several machines to improve horizontal scalability using distributed frameworks. The major concerns of the service layer are ensuring low latencies and high throughput by using distributed frameworks.

3.3 Implementation Workflow

Referring Fig. 1.

- Mapper Phase

Step 1: Miner Mapper

Working: Miner mapper is used to bifurcating the available blockchain miners into n shards.
Input: Each miner is assigned a unique number and M_id indicates the miner id.
Output: M_id indicates the miner id and S_id for shard id.

Step 2: Transaction Mapper

Working: Bifurcate the blockchain transactions into k shards.
Input: All transactions in the transaction pool are referenced using two tuples < tx_id, tx > where tx_id denotes unique transaction ID and tx indicates transaction parameters of blockchain.
Output: The transaction mapper outputs a two-tuple output < tx, S_id > .

The accumulated transactions are divided into k splits, and the transaction mapper outputs a key as transaction and S_id the shard id to which the transaction belongs.

Step 3: Transaction-Miner Mapper

Working: 'k%n' transaction shards are mapped to each miner shard. If $(k = n)$, then one-one mapping will take place between miner shards and transaction shards.
Input: 'k' transaction shards and n miner shards.
Output: The transaction-miner mapper outputs a two tuple <S_id, <List(txns), List(Miners) $\gg$ S_id which indicates miner-transaction shard, List(txns) indicates the list of transaction shards in a miner-transaction shard and List(Miners) indicates the list of miner shards in miner-transaction shard.

Step 4: Reducer Phase

Working: used to apply consensus(aggregation) on the individual mappers. The number of reducers is decided using Eq. (2).
Input: Receives two-tuple <S_id, <List(Txns), List(Miners) $\gg$ List(txns) indicates the list of transaction shards in a miner-transaction shard list(miners) which indicates the list of miner shards in miner-transaction shard.

Output: Miner transaction shards apply consensus algorithms on the transaction shards and propose new blocks. The output of this layer is a two tuple <B_i, S_id> where 'B_i' represents the block proposed by shard *i*.

Algorithm-

```
Procedure reducer (List (<S_id, <List(Txns),
List(Miners)>>) returns list (<Block B_i, S_id,
List(Miners)>)
Foreach shard i <-0 to i<-n-1 with step=1 do
  Step 1: ∀ miners m ∈ shard i CreateBlock(List(Txns))
  Step 2: ∃miner m ∈ shard i ProposesBlock(List(Txns))
          returns B_i
  Step 3: d<- ∀ miners m ∈ shard i perform
          consensusAlgorithm(Block B_i)
          If(d==false) then
            DiscardBlock(Block B_i)
End Foreach
```

Step 5: Final Consensus Phase

Working (*Referring* Fig. 2).

The consensus algorithms used in this phase are Byzantine Fault Tolerance Algorithm and its variants (based on the criteria mentioned in Table 1). Byzantine fault

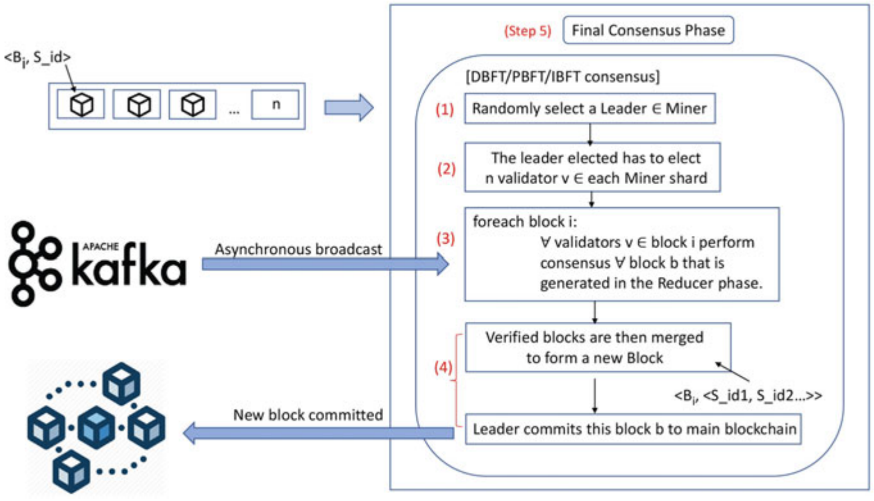
<B_i, S_id>
n
...
(Step 5) Final Consensus Phase
[DBFT/PBFT/IBFT consensus]
(1) Randomly select a Leader ∈ Miner
(2) The leader elected has to elect n validator v ∈ each Miner shard
(3) foreach block i: ∀ validators v ∈ block i perform consensus ∀ block b that is generated in the Reducer phase.
(4) Verified blocks are then merged to form a new Block
<B_i, <S_id1, S_id2...>>
Leader commits this block b to main blockchain
APACHE kafka
Asynchronous broadcast
New block committed

Fig. 2 Detailed implementation workflow architecture of the final consensus phase

Table 1 Comparison among consensus algorithms

	Proof of work	Proof of stake	Practical BFT	Delegated BFT	Federated BFT	Istanbul BFT	Delegated POS
Description	Node has to prove that it has performed amount of work	Node has to stake the ownership of the virtual currency	Node are sequentially ordered with one as the leader and the others as backup nodes	Delegates are elected, and then a speaker elected from them proposes a block to all the other delegates	Variants of the BFT by making them open-ended with respect to node participation	A proposer proposes a block and validators agree and broadcast their decision (geth implementation)	Elects the set of proposers and validators based upon stake and punish on malicious behavior
Used by	Bitcoin Blockchain, Ethereum (EthHash)	Cardano	Zilliqa, Hyperledger fabric	Neo	Ripple, Stellar	–	EOS
Token needed	Yes	Yes	No	No	No	No	Yes
Throughput	~7 txns/s	~10–15 txn s/s	~10,000 txns/s	~1000 txns/s	1500–2000 txns/s	400–1200 txns/s	50–250 txns/s
No. of commits	6 confirmations	2/3rd confirmations	f + 1 (f is no. of faulty node)	2/3rd of delegates agree	$2/3^{rd}$ confirmations	2f + 1 (f is no. of faulty node)	$2/3^{rd}$ confirmations
Possible attacks	51% attack	Nothing at stake attack Sybil attack	Nodes increase then msg count increases	>33 % dishonest delegates	Malicious validators	If honest nodes are less than dishonest	Low voter turnout attack
Scalability of peer network	High	High	Low	Low	High	Low	High
Cost of participation	Yes	Yes	No	No	No	No	Yes
Transaction finality	Probabilistic	Probabilistic	Immediate	Immediate	Immediate	Immediate	Probabilistic

tolerance algorithms provide better throughput compared to PoW and other consensus algorithms when there are limited number of participants participating in the consensus phase. The cost of participation in Byzantine fault tolerance and variants consensus is less compared to PoW and PoS, thus enabling lucid addition of nodes into the consensus phase. In BFT and its variants, only a part of the miners participate in the consensus rather than the whole, thus reaching faster consensus and ameliorating the validation of unconfirmed transactions in the transaction pool. They provide faster consensus in limited number of nodes compared to other consensus models. From the list(miners) which we receive from the reducer phase. We select a leader from the global miner list who samples h miners from each shard and form a group of h*n. This group formed needs to perform the final consensus on the blocks B0 to Bn-1. All the members of the group that has been formed can communicate with each other asynchronously using Apache Kafka's publisher subscriber model and can exchange data between each other seamlessly. Blocks which are of the form <Bi, <S_id1, S_id2, … ≫ indicate that they have received consensus from all the validators, and the block is committed to the main blockchain.

Algorithm-

```
STEP1: Randomly select a leader $\in$ globalList(Miners)
STEP2: The leader elected has to elect h miners v$\in$ shard$_i$
using randomly to form a group of h*n members known as
validators Set V
STEP3: Foreach Block i <-0 to i<-n-1 with step =1 do
         STEP3.1: d$_i$<- $\forall$ validators $\in$ V$_i$ perform
                 consensusAlgorithm(Block B$_i$)
        END FOREACH
STEP4: Foreach Block i <-0 to i<-n-1 with step =1 do
         If (d$_i$==true) then
           commit the blocks B$_i$ which are of the form
           <B$_i$, <S_id$_1$, S_id$_2$, S_id$_3$..., S_id$_n$> to the
         mainchain.
         Else
         DiscardBlock(Block B$_i$)
      END Foreach
```

3.4 Estimating the Optimal Values of Mappers and Reducers

There are several parameters in MapReduce that need to configure to get better performance than the baseline performance that MapReduce provides. Tuning of hyperparameters is necessary to get optimal performance [19].

Determine the Number of Mappers

$$M = (P - I) * K \tag{1}$$

- 'M' denotes the number of mappers.
- 'P' denotes the number of physical cores.
- 'I' denotes the number of reserved cores.
- 'K' denotes the CPU hyperthreading factor which ranges from [0.95 to 1.75].

Upper limit on the number of mappers is $M = F/B$.

where $F =$ input file size supplied to the cluster
$B =$ block size that is used by the cluster.

Determine the Number of Reducers

$$R = K * (\text{number of nodes} * \text{mapred.reduce.parallel.copies}) \tag{2}$$

- 'K' denotes the CPU hyperthreading factor which ranges from [0.95 to 1.75].
- 'Mapred.reduce.parallel.copies' is the maximum number of reducers that can execute in parallel.
- 'R' denotes the number of reducers.

The upper and lower limits on the number of reducers should be in the range [C/2, 2 * C] wherein C denotes the number of CPU cores. Apply 2/3 mapper technique which states that the number of reducers should be 2/3 that of the number of mappers estimated.

These formulas above have been used to determine the values in our proposed blockchain solution. These starting values obtained by these formulas act as a starting point to bifurcate the mining and transaction pool of our blockchain solution. These parameters help to determine the processing power required to run this architecture, and these hyper parameters can be fine-tuned over time.

3.5 Comparative Analysis Among Consensus Algorithms

This section provides a comparative analysis between consensus algorithms based on the criteria mentioned in Table 1 [22]

In our solution, the type of consensus algorithm in the intershard and intrashard consensus phase affects the performance of our system. BFTs are apt for permissioned blockchains, whereas the rest are good for public chains. There is a clear trade-off between scalability of consensus algorithms and time consumed for reaching consensus. Hence, BFTs are less scalable compared to the rest. Apart from PoW, PoS and Dpos, all the other consensus algorithms have the capability to revert the transaction from the confirmed block. Transaction finality is immediate in BFTs, whereas the others have probabilistic finality. By reducing the participation of peer nodes in consensus, we can certainly boost the transaction rate in a blockchain architecture.

Using the above parameters, we choose in the consensus algorithms for step 4 and step 5.

3.6 Performance Analysis Between the Proposed Solution and Existing Solutions

- *Block Commit Time*: required average time to commit the block to the main chain since it was created. Our solution aims to optimize this time by sharding the transaction pool compared to other solution by using Hadoop lambda architecture (Table 2).
- *Transaction Confirmation Time*: average time needed for a transaction to be confirmed into a mined block. Our solution uses faster consensus mechanism at intershard and intrashard consensus layers compared to other solutions.
- *Mempool Size*: Our framework can handle differential Mempool loads with the help of the lambda architecture compared to the other existing solutions.

Table 2 Comparative analysis between the proposed solution and existing solutions using performance metrics

Criteria	Proposed solution	Zilliqa and similar
Block commit time	Low	Moderate
Transaction confirmation time (depends on consensus)	Low to moderate	Moderate
Mempool size	High	Moderate
Network down time	Low	Low to moderate
Intershard communication	Moderate	Low
Transaction throughput (TPS)	High	High

- *Network Down Time*: Recovery of nodes in our solution is possible due high availability and replication features of Hadoop which are lacking in Zilliqa.
- *Intershard Communication*: Intershard communication in our solution is easy compared to other solutions because of Apache Kafka which provides seamless interaction among the shards which is hindrance to other solutions.

4 Conclusion

We have looked at an interesting way of how we can leverage big data concepts to resolve the scalability concern in distributed ledgers. For the past decade, the amount of reliance of user and application generated data have increased manifold. Big data technology has enabled us to scale at par with the increase in demand. Blockchain is now being viewed as a potential technology of the future, replacing the current centralized architectures.

In this paper, by focusing on the scalability aspect in blockchain, we have addressed one of the potential points of hindrance for one of the most important technologies of the future. The framework aims to integrate the well-established big data *sharding* frameworks with blockchain to ameliorate the scalability of blockchains. The paper proposed an initial overview of our research in the blockchain field and a framework with native big data *sharding* framework called *MapReduce* and aims to use an integrated *MapReduce* framework like Apache Spark for future work.

We aspire to integrate the present big data solutions with the present blockchain solutions.

References

1. Buterin V (2015) Notes on scalable blockchain protocols
2. Satoshi N, Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. Bitcoin
3. Buterin V (2013) Ethereum white paper: a next generation smart contract & decentralized application platform. Ethereum
4. Buterin V (2014) Proof of stake: how i learned to love weak subjectivity
5. The raiden network (2018) https://raiden.network/
6. Poon J, Buterin V (2017) Plasma: scalable autonomous smart contracts. White Paper
7. Teutsch J, Reitwießner C (2017) A scalable verification solution for blockchains. https://People.Cs.Uchicago…, p 50
8. Donald Miner AS (2014) Map reduce design patterns
9. Reilly O, Seu M (2012) Hadoop, the definitive guide. Online
10. Knox Gateway (2019) https://knox.apache.org/
11. Apache Ranger (2019) https://ranger.apache.org/
12. Abraham I, Malkhi D (2017) The Blockchain Consensus Layer and BFT. Bull EATCS
13. Wood G (2017) Polkadot: vision for a heterogeneous multi-chain framework. Whitepaper
14. Kwon J, Buchman E (2018) A network of distributed ledgers
15. Kwon J (2014) TenderMint : consensus without mining

16. Team TZ (2017) Zilliqa technical whitepaper. Zilliqa
17. Using Merklix tree to shard block validation | Deadalnix's den (2016) https://www.deadalnix.me/2016/11/06/using-merklix-tree-to-shard-block-validation/
18. Charles Hoskinson (2017) Why are we building Cardano
19. Tannir K (2014) Optimizing Hadoop for MapReduce. Packt Publishing
20. Kiran M, Murphy P, Monga I, Dugan J, Baveja SS (2015) Lambda architecture for cost-effective batch and speed big data processing. In: Proceedings—2015 IEEE International Conference on Big Data, IEEE Big Data 2015
21. Shapira G, Narkhede N, Palino T (2017) Kafka: the definitive guide—O'Reilly Media. O'Reilly Media
22. Baliga A (2017) Understanding blockchain consensus models. Whitepaper