Final Report
Blue-Collar Node : A Sharding Approach for Optimizing Storage on Blockchains

1st Chawla
Computer Science
Arizona State University
nchawla3@asu.edu

2nd Saikia
Computer Engineering
Arizona State University
csaikia@asu.edu

***Abstract*—The scalability of blockchain is a primary and urgent concern. The current popular blockchains have fundamental bottlenecks which limit their ability to have a higher throughput and a lower latency. One of the bottlenecks is the storage requirements of the current blockchains. All the full nodes and miners in a blockchain are required to store the complete blockchains and it grows every time more transactions are added. The network relies mainly on commodity hardware for block propagation and transaction verification. With the increase in the size of blockchains, storing the entire blockchain on voluntary full nodes becomes impractical. Our project focuses at sharding this blockchain and storing it in several nodes. There is a two-fold advantage of this, transactions can be verified in parallel by different nodes and consensus can be achieved in a faster way increasing the throughput of the network, and the storage requirement of the full node will be decreased substantially. That directly will prevent the blockchain from getting centralized towards supercomputers. However, in this project we will focus only on the storage part of the problem. Executing transaction in parallel will be a part of the future work.**

***Index Terms*—blockchain, storage, sharding, blue-collar, decentralized**

I. Introduction

In 2009, Bitcoin [8] introduced the first blockchain which is an append only database that stores transactions. Blockchain maintains the distributed database in a decentralized network aiming to solve the Byzantine Generals Problem [13]. Byzantine Generals Problem when extrapolated to the banking system i.e to digital money translates to double spending problem, where since there is no material money a malicious node can spend the money twice, and since there is no central leader to check the money can be spent.Bitcoin solves this problem by distributing the ledger to every node in the network and cryptographically securing it making it immutable. Special nodes called miners try to include the transactions into blocks by solving a puzzle. This mechanism is called Proof of Work. Each miner then tries to add this block to the distributed replicated ledger and after the block is added each node on the network updates their ledger.

A. Problems

Satoshi Nakamoto decided a block interval of 10 minutes between adding 2 blocks to the chain. The block size was limited to 1MB from ”unlimited number of transaction” because of a DoS attack that happened in 2010. Since, the block size is capped and the time interval is constant, the number of transactions that can be dealt with are limited to 3-7 transactions per second. Also, since the ledger is duplicated on each node on the network, as the ledger grows the storage requirement of the node increases.

II. Background And Related Work

A. Background

Blockchain is a decentralized, tamper-proof digital ledger technology that is transforming transaction prospects for many industries. In addition to being the foundation of cryptocurrencies like Bitcoin,Ethereum,Dash,etc the blockchain structure, which consists of linked blocks of information, allows for direct transactions across a network of computers without need for a central authority. The Dash network consists of four kind of nodes: **Masternodes**, **full nodes**, **light-weight nodes** and **miners**. Nodes save the complete blockchain and have two main functions:

(a) to validate transactions
(b) to propagate transactions and blocks in the network

Although full nodes validate transactions by checking their inputs and outputs, light clients verify transactions by SPV, or by requesting random chunks of a block and using the merkle root to verify each chunk. This verification technique relies on trust on a full node or the node sending data. This can be dangerous, because a malicious node can withhold small amount of data and can get unnoticed by many light clients. The network becomes much more trustworthy if the number of full nodes are a majority since data is always verified. As the blockchain technology is being adopted more, the storage requirement is increasing for each individual node. This increase in infrastructure requirement forces volunteer nodes to move to light-clients. This harms the network in 2 ways :

(a) The network gets more centralized to people who can afford supercomputers
(b) The trust on the network weakens as most transactions are not completely validated by full nodes

In this project, we want to introduce a new kind of node that saves only a part of the blockchain thus reducing the storage requirement and also validates each transaction that is a part of the block. This will introduce the following advantages to the network:

(a) The trust on the network will be high as the nodes will validate each transaction fully.
(b) Randomizing block storage in such a way that transactions are fully validated by a small part of the network leading to multiple transactions being validated in parallel.

We will be focusing only on the first advantage as a part of this project, the second would remain a future work.

B. Related Work

Elastico [5] also attempted sharding by adding the concept of committees. The complete protocol was revamped by dividing it into 2 rounds. The first round involves proof of work to choose committees for each node in every block mining cycle. Once the committees are assigned, nodes use Practical Byzantine Fault Tolerance [**?**] in order to form a block based on the transactions that they are to verify. Once transactions are verified, the partial block is pushed forward to a final committee which finalizes and mines the new complete block. The throughput of elastic [5] increases linearly with number of committees in the network. This can be shown in the 1.

”Sharding FAQ [4] gives a good glimpse of the problem we are trying to solve with techniques that will come in handy for us to make our solution better.The difference in the approach is that [4] tries to scale the throughput of the blockchain while we focus on the storage optimization of the system.

”A note on data availability and erasure coding” [3] is another blog by the founder of Ethereum [9] [10] which touches our areas of concerns. It talks about light clients and how their lack of validation techniques might be harmful for the network. It exposes a loop-hole in the system which we want to remediate by the creation of ”blue-collar” nodes.

III. Solution: Work In Progress

A. Components

The key component of our system will be what we are calling a **”Blue-collar node”**. This node will be a combination of a light and full-node. This node will be required to store the sharded blockchain and the parity bits such that if needed it can re-create the data from a connected full node. Creation of shards is done by Erasure Code [11] and currently Reed-Somon algorithm [12].

Masternodes are special nodes present in the dash network that have special characteristics. These are proof of service nodes that are paid a mining awards in order to provide the service. These masternodes add reliability to the network and can be trusted upon to store the entire blockchain. Masternodes have high number of connections because they are required to run at 1GBps bandwidth and most nodes are connected to at least one Masternode. We will be utilizing this property of masternodes to recreate the data in the newly formed ”Blue-Collar node” that when needs to find a missing transaction pings a Masternode.

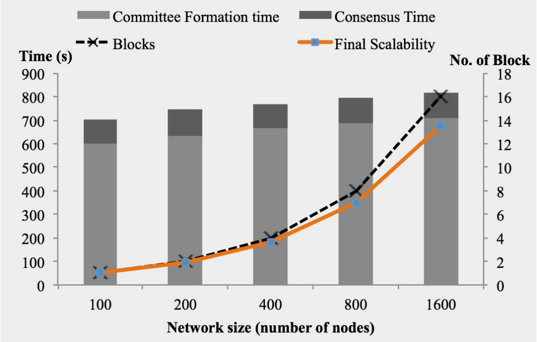

Fig. 1. Elastico Sharding

However, for open blockchains Kademlia DHT will be used with it’s property of XOR to find the closest neighbour that has the data.

B. Architecture

Dash [1] has a unique privacy-centric infrastructure in place. It employs 4 kind of nodes:

(a) **Masternode**: These are special nodes in the dash network that require 1000 dash to be active on the network. These nodes add special features to the network such as **voting** for the proposals that are active on the network, **high-bandwidth** for better relay, **better hardware** for faster transaction processing, etc
(b) **Full-nodes** These nodes are same as master nodes but are voluntary nodes and are a part of network without the need of having a 1000 dash in their wallets.
(c) **SPV/Mobile Clients** These are basically mobile clients that don’t download the blockchain but verify transactions by just combining the hashes of merkle tree.
(d) **Miners** These are nodes that help adding new blocks to the blockchain. They require to have a compute intensive hardware in order to find the next random number(nonce) that satisfies the difficulty of the network.

All nodes apart from the SPV wallets are required to store the entire blockchain along with indexed leveldb/berkeley db databases in order to verify the transactions.

C. Challenges

In the methodology described above there are certain challenges that are not yet handled. Some of them are listed below:

1) The data distribution has to be randomized in such a way that a malicious intentions can’t get hold of data that can

be completely lost. In such a case, the contingency would be to recover the data from a full node and distribute it again over the network among peers but would require a lot of data movement.

2) The data has to be distributed in such a way that in case some nodes fail, the re-distribution of data does not cause much data movement and doesn't occupy bandwidth that could be used for block propagation.

D. Theory

1) Erasure Code

In distributed storage systems, failures are inevitable. Therefore, it is very important to design these systems with some high availability technique. One way to do it is by introducing redundancy. One of the well known techniques for protecting data is erasure coding. It is a widely used and accepted technology used in communication systems and more recently, in storage systems. In storage systems, erasure codes add redundancy to the system in order to tolerate failures. Erasure codes vary from simple coding techniques, such as a full replication of the data, such as RAID-1, where each byte of the data is stored on two disks. However, this increases the storage cost. More complex erasure codes, such as Reed-Solomon codes, exhibit fault tolerance with less extra storage compared to RAID. Thus, this is a more cost effective approach.

Let's assume that our storage system consists of n disks. These disks can be partitioned into k disks so that these disks can hold the user data. Thus the remaining m=n-k disks hold the coding data. The encoding and decoding techniques are explained in Figure X.

In the encoding process, the contents of the k data disks are used to calculate the contents of the m coding disks. Thus, this kind of storage system can handle failure of up to m disks. When any number (up to m) disks fail, then all the data is decoded from the up and running disks. The simplest erasure codes assume that each disk consists of one w-bit word. Optimal erasure codes are maximum distance separable codes. Optimal codes are expensive in terms of memory usage and CPU time for large value of n. The kind of erasure coding used in this experiment are Reed-Solomon codes. Reed-Solomon codes are MDS codes. For disks where $n <= (2^w)$, Reed-Solomon codes are usually used. For example, for storage systems which contain less than or equal to 256 disks, Reed-Solomon is defined for it. Reed-Solomon codes operate on a block of data treated as a set of finite field elements called symbols. For example, a block of 4096 bytes can be considered as a set of 2731 12 bit symbols. Here, each symbol is a finite field element of $GF(2^12)$ and the last symbol is padded with four 0 bits. There are different ways to define the a_ij coefficients and one of them is the "Cauchy" construction. A distinct number n is chosen in $GF(2^w)$ and then the disks are partitioned into two sets X and Y. X set has m elements and Y has k elements. Then,

$a_ij = 1/(x_i XOR y_i)$, where the arithmetic is over $GF(2^w)$.

Fig. 2. Erasure Encoding and Decoding

Any value of k and m generates result. However, they are expensive since complexity of multiplication in a Galois Field [16] is more expensive then XOR.

Encoding and decoding is shown in the 2

2) Raptor Code [18] : Future Work

Raptor code are rateless erasure code as opposed to Erasure code whose rate is determined by values k and n described above. The rateless here means that limitless sequence of encoding symbols can be generated from source symbols and can be recovered by any subset (k symbols) of encoding symbols that are only slightly larger than the original symbols. These have linear encoding and decoding time as opposed to Erasure code that exhibit a quadratic time. Since these codes are faster in decoding, it has a potential use case of decoding large files that are continuously broadcasted to a set of receivers. Raptor codes are heavily covered in patents as of now, therefore currently Erasure codes have been used for this project.

3) Distributed Hash Table: Kademlia : Work In Progress

We are in the process of using openDHT [17], which is an open source library developed by UC Berkeley for implementing Distributed Hash Tables. We have chosen Kademlia over other hash table structures like chord because Kademlia contacts only $O(logn)$ nodes in order to retrieve the location of the information.

Since we are able to implement erasure codes, DHT will help us dissipate that information, update the table. The decoding function is already written. It is not yet integrated with the original code, because in order to decode we would require the location of the data.

E. Implementation

The implementation is done in 3 parts:

1) The emulator

Since this is a complex problem to be solved on a decentralized distributed system. In order to achieve and test the desired results, an emulator was set up that automates the following process:

(a) Building dependencies on each node.
(b) Building dash static binaries to form light clients.
(c) Building docker clients with running docker clients.
(d) Forming a network of nodes with random number of connections with minimum and maximum set.

2) Automating Dash build with Docker

In this phase we have set up a docker container with all packages statically built in order to optimize space storage inside the container. We have used "Ubuntu" as the base image for docker. The dash build is not one of the release versions but is taken from the "development" branch in order for us to contribute directly. A combination of shell, python and docker is used to automate the installation of dependencies along with the original dash client.

3) Setting up Regression Test Network

In this phase we have used the same automation tool and cloned many docker clients with different ports. The dash clients in ***-regtest*** mode do not automatically discover new nodes even with ***-discovery=1*** flag because the *peers.dat* file that has a regular feed of peers does not have any seed from the current local network. In order to rebut that we randomly assigned peers to each node by using the algorithm Algorithm 1:

Algorithm 1 Connect Nodes

```
while numberOfConnections for i $<$ minConnections[i] and count $<$ 10*minConnections[i] do
    index $\leftarrow$ random(0,numOfNodes*100)% (numOfNodes - 1);
    candidaePeer $\leftarrow$ index;
    if candidaePeer $==$ i then
        print ”Can’t connect to itself”
    else if nodesConnections[i] $>=$ maxConnections[candidatePeer] then
        print ”Node already has maximum connections”
    else if candidatePeer in nodesConnections[i] then
        print ”Node already has maximum connections”
    else
        append candidatePeer to nodesConnections[i]
        append i to candidatePeer
        add peer candidatePeer to i
    end if
end while
```

4) Adding sharding techniques to the dash original repository and integrating it our docker-dash network

In order to create erasure codes we are using the write-to-disk methodology in the original dash code. The library Jerasure [15] has been used in order to create erasure codes through different encoding algorithms with different values of ***k*** and ***m*** in order to find an optimal way of distributing data among the nodes.

Also, we have changed the parameter ***MAX_BLOCK_FILE*** to 256K instead of 128 MiB for the ease of testing.

We choose the latest file that has been recently created and is full. We divide that information into $k+m$ files. These m files are the ones that we can lose and still be able to regenerate the data. In addition to these files, a small 4K meta file is created that consists the encoding parameters.

This meta files must be present on all the nodes that are using the sharded blockchain in order to recreate the data when needed for verification.

IV. Results

The following accomplishments have been made during the course of this project:

1) Automation scripts for a dash build have been made.
2) These automation scripts help in building an emulator, our own testnet to test theories that need to be researched.
3) Erasure encoding using Reed Solomon has been successfully implemented in the dash original source code.
4) Division of data file into respective code along with redundancy have been made so that loss of several files could still regenerate the data.
5) Decoding script is built but not integrated because of the complications we are facing in implementing Kademlia into the network.

References

[1] Duffield, Evan, and Daniel Diaz. ”Dash: A privacy-centric cryptocurrency.” (2014).

[2] Weil, S. A., Brandt, S. A., Miller, E. L., & Maltzahn, C. (2006, November). CRUSH: Controlled, scalable, decentralized placement of replicated data. In Proceedings of the 2006 ACM/IEEE conference on Supercomputing (p. 122). ACM.

[3] Buterin, Vitalik. ”A note on data availability and erasure coding” Github. https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-coding Accessed 2nd January, 2018 .

[4] Zamyatin, Alexie. ”Sharding FAQ” Github. https://github.com/ethereum/wiki/wiki/Sharding-FAQ Accessed 2nd January, 2018

[5] Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., & Saxena, P. (2016, October). A secure sharding protocol for open blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 17-30). ACM.

[6] Zastrin. https://www.zastrin.com/courses/1/lessons/2-3

[7] Honicky, R. J., & Miller, E. L. (2004, April). Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. In Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International (p. 96). IEEE.

[8] Nakamoto, Satoshi. ”Bitcoin: A peer-to-peer electronic cash system.” (2008).

[9] Wood, Gavin. ”Ethereum: A secure decentralised generalised transaction ledger.” Ethereum Project Yellow Paper 151 (2014): 1-32.

[10] Buterin, Vitalik. ”Ethereum white paper.” GitHub repository (2013).

[11] Dimakis, A. G., Godfrey, P. B., Wu, Y., Wainwright, M. J., Ramchandran, K. (2010). Network coding for distributed storage systems. IEEE transactions on information theory, 56(9), 4539-4551.

[12] Reed, I. S., & Solomon, G. (1960). Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics, 8(2), 300-304.

[13] Lamport, L., Shostak, R., Pease, M. (1982). The Byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS), 4(3), 382-401.

[14] Hafner, James Lee. ”WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems.” FAST. Vol. 5. 2005.

[15] Plank, James S., Scott Simmerman, and Catherine D. Schuman. ”Jerasure: A library in C/C++ facilitating erasure coding for storage applications-Version 1.2.” University of Tennessee, Tech. Rep. CS-08-627 23 (2008).

[16] http://jerasure.org/gf-complete-1.02/

[17] Rhea, Sean, et al. ”OpenDHT: a public DHT service and its uses.” ACM SIGCOMM Computer Communication Review. Vol. 35. No. 4. ACM, 2005.

[18] Shokrollahi, Amin. ”Raptor codes.” IEEE transactions on information theory 52.6 (2006): 2551-2567.