Midterm Report

Stacey Dale Robert Pomichter Shelby Menown
Hannah Moss Patrick Bauer

March 2021

1 Introduction

Blockchain is a technology which first emerged in the academic community in 1991 when Stuart Haber and W. Scott Stornetta conceived of a solution to cryptographically secure data. Specifically, Haber and Stornetta developed a chaining technique which maintained the integrity of records.

It was not until 2009 when a person or group known as Satoshi Nakamoto published the first whitepaper on blockchain; as it was implemented by Bitcoin, that blockchain technology became more widely recognized. When Nakamoto published the whitepaper, Blockchain was used as a ledger system for cryptocurrency. Nakamoto and Bitcoin were motivated by the idea of a completely decentralized form of currency where the complete history of transactions is distributed to everyone participating in the system. To this day, the focus of blockchain technology is primarily the decentralization of the system and the integrity of records[4].

Blockchain works by collating transactional records in structures referred to as blocks. Each block is constructed and appended to a chain of older blocks, which themselves were mined and appended to the chain previously. The block is constructed and then distributed to the network of computers referred to as nodes, where participants attempt to mine the block. After the block is mined successfully by one of the nodes in the network, the other nodes validate the newly mined block by verifying the block for correctness.

The process of mining, validating, and appending the block to the chain is at the heart of blockchain technology. Mining is a process which goes by other names depending on the consensus algorithm utilized in the technology. In the case of Bitcoin and other Proof-of-Work models, mining is a race between all nodes on the network to solve a computationally difficult problem. The first node to successfully solve the computationally taxing problem is said to mine the block.

Nodes on the network attempting to mine the block are referred to as miners. In the case of Bitcoin, miners are incentivized to mine blocks because a successfully mined block yields a reward in the form of cryptocurrency. The

complexity of the mathematical problem is periodically adjusted to ensure the blocks are not exclusively mined by one node on the network. This is the tenuous balance maintained between the blockchain and the miners. As the focus of blockchain technology is decentralization, it is imperative that no one node on the network obtains more than a majority share of all blocks in the chain[4].

There are many other consensus algorithms which all attempt to maintain the fundamental properties of the blockchain technology–integrity and decentralization. These other consensus algorithms include Proof-of-Stake, Proof-of-Burn, Proof-of-Capacity and many others which differ in terms of implementation and by approach to perceived threats on the blockchain[1]. The Proof-of-Work consensus algorithm is the primary focus of this project.

2 Overview

In this project, we implement an educational blockchain in three languages, C++, Java, and Haskell, first sequentially and then concurrently. We chose these three languages because each language uses threads differently. Due to the abstraction the JVM performs, Java threads can run differently depending on the underlying operating system. The user may not be aware of how the JVM is ensuring correct synchronization and concurrency. C++ is considered to have more efficient threads because they are not abstracted through any JVM. Haskell uses fine-grained threads, similarly to C++. However, Haskell has added efficiency in other mathematical calculations, thus making it a valid educational choice for a concurrent blockchain. Choosing an educational blockchain over common blockchain algorithms was an important part of our planning stage, ultimately allowing us to maintain our early decision to use the three languages discussed above to compare concurrent blockchain efficiency.

Initially, we expected to implement some form of Ethereum blockchain using concurrent smart contracts. This idea spawned from current discourse in the field of blockchain and concurrency. However, as we began to research Ethereum blockchain, with reference to the Ethereum whitepaper, we realized that 1) implementing Ethereum blockchain in general would be labor-intensive and 2) using our three chosen languages would not be the best approach to implementing Ethereum. Based on these two realizations, we were able to conclude that adding concurrency to Ethereum blockchain implemented in three different languages was far outside the scope of a semester-long, undergraduate project. Indeed, even becoming familiar with concurrency in Haskell and C++ was going to be a stretch for our team, since we started the semester only with a basic understanding of concurrency in Java.

Thus, we faced a decision: discard our plan to use and compare concurrent blockchain across different languages or change the type of blockchain we would create. We chose the latter after finding a reasonable algorithm for a sequential blockchain in the C language. Our task then became translating the algorithm into each language to create single-threaded blockchains and implementing comparable concurrency across the three different blockchains. By

comparable, we mean using the most logically consistent algorithm we could implement across languages. Logical consistency is quintessential to our analysis; if one blockchain implements a more efficient concurrent algorithm, then the results cannot be compared to our expectations. If, however, the same concurrent algorithm is applied to each blockchain and one blockchain performs much better due to the types of threads it uses and the efficiency of the language, then we can analyze the results in accordance with our expectations.

Since we had decided to use an educational blockchain that does not implement Smart Contracts, we needed to determine which part of the blockchain we would make concurrent. Because the educational blockchain structure we chose uses a Proof-of-Work consensus similar to Bitcoin, we realized that we could use concurrency to solve the puzzle. In Bitcoin, this puzzle varies in difficulty with each run, with the time to solve the puzzle averaging ten minutes per solve. On any given puzzle, though, the difficulty could be greater or less than a ten minute solve. Testing the efficiency of concurrency on an ever-changing difficulty would be complicated, and the results would only be questionably reliable. Thus, our blockchain implementation does not change the difficulty of the puzzle across different tests. Additionally, each blockchain uses the same input file, so the only difference across tests is the timestamp. This is a reasonable difference that should not alter our results or subsequent analysis and conclusion.

Each blockchain implementation is merely academic. It does not support peer-to-peer networks. Rather, one machine creates the blockchain and adds blocks (nodes) with only one transaction (line from the text file). Our focus is 1) to test the efficiency of building the block chain without and with concurrency in one language and 2) to compare the level of efficiency in one language to others. Because of this, our actual code cannot be considered for implementation into current blockchain models. However, current discourse in blockchain is honing in on concurrency, and specifically how concurrency can be added into certain parts of the transaction process to improve efficiency. Our project is yet another exploration into how concurrency can fit into the blockchain process.

3 Research

We spent the first two weeks of our project conducting and combining research via a tool called Zotero. Upon choosing a far-too-broad topic, applications of concurrent blockchain, we dove into foundational research to answer a few broad questions: What is block chain, really, and What is the current discourse on concurrency in block chain?

These two questions led us to explore all aspects of blockchain. We first discovered "A survey of blockchain consensus algorithms performance evaluation criteria", a journal article by Seyed Bamakan, Amirhossein Motavali, and Alireza Bondarti published in Expert Systems with Applications [1]. This article not only provides an in-depth explanation of blockchain, but also details twelve types of consensus algorithms, including the algorithm we ultimately chose to implement: Proof of Work.

We continued to research blockchain, now focusing on concurrency. We found Laxmi Kadariya's thesis on concurrent block chain, "Concurrency in Blockchain Based Smartpool with Transactional Memory" [4]. In his thesis, Kadariya covers everything from an explanation of features and aspects of blockchain to a description of concurrent programming to provide a background for the proposal of a non-blocking concurrency mechanism for block verification within a decentralized mining pool.

Kadariya focuses primarily on Ethereum blockchain, which then led us to consider Zachary Painter, Victor Cook, Damian Dechev, and Pradheep Gayan's article "Parallel Hash-Mark-Set on the Ethereum Blockchain" [6]. This article provides a concurrent solution to the bottleneck caused by the sequential implementation of the Hash-Mark Set algorithm that allows peers to access a READ-UNCOMMITTED view of the blockchain network (that is, the network includes nodes that are not yet published). Because this article primarily looks at Ethereum blockchain, we began considering trying to implement the proposed concurrency algorithm.

We began researching Ethereum blockchain via the Ethereum Whitepapers [2]. The Whitepapers are thorough, not only providing plain language explanations of implementation, but also clear suggestions on how to implement. We noted quickly that 1) Solidity is a language made specifically for Ethereum, and thus if we actually planned to make a standalone Ethereum blockchain, we should use that language, and 2) implementing an Ethereum blockchain was a massive undertaking. We began researching other options for implementing concurrency in blockchain.

Our research led us to simple blockchain implementations in C and Java [5, 3]. Both of these implementations are academic in nature; neither is interested in a peer-to-peer network of nodes, but rather in the method of creating a blockchain via hashing and mining. Both usea Proof-of-Work consensus. While neither implement concurrent blockchain, we were able to determine which parts of the process lend themselves to concurrency.

And thus, the real work began.

4 Expectations

The Proof-of-Work consensus algorithm forces nodes to find target hash values within a limited range. The narrower the range of the hashed value, the more computationally taxing the problem becomes. Since hash values generated by the SHA-256 cryptographic hashing algorithm vary drastically between even small changes made on the input value, nodes on the network must use a brute force approach to finding the target value[5].

The brute force approach involves iteratively generating hash values by changing the input value incrementally after each failed attempt using the nonce value. This approach is almost random, as no node on the network has the ability to determine which input value will successfully generate the target hash value. This process is the most laborious process in the blockchain and the

most time consuming[3].

By increasing the number of threads concurrently generating hash values, the expectation is naturally that there will be on average a decrease in the time between successfully mined blocks. This approach to increasing the rate by which blocks are mined is analogous to purchasing multiple raffle tickets. In other words, it is likely that someone who purchased 50 tickets when entering a raffle has a probabilistic advantage when compared to another individual who only purchased one ticket when entering the same raffle. However, due to the random nature of the raffle, and similarly the Proof-of-Work algorithm, it is still possible for someone who purchased fewer tickets to win the raffle. Likewise, a node with fewer working threads could still find the target hash before a node with more working threads.

Our expectation is to observe an overall increase in the rate of successfully mined blocks by a node with more threads than a node with fewer threads. We also expect that occasionally the node with fewer threads will successfully mine a block before a node with more threads. We are especially interested to observe the relationship between the number of worker threads and the overall rate of successfully mining blocks. Furthermore, across languages, we expect concurrent implementations in C++ and Haskell to perform better than the concurrent implementation in Java, since C++ and Haskell use low-level threads.

5 Methodology

Describe the steps we have taken and plan to take to complete this project.

6 Results

Analyze the results of testing and compare to our expectations.

7 Conclusion

Draw conclusions based on analysis. Explain how our project fits into the concurrency discourse in block chain research.

References

[1] Seyed Mojtaba Hosseini Bamakan, Amirhossein Motavali, and Alireza Babaei Bondarti. A survey of blockchain consensus algorithms performance evaluation criteria. *Expert Systems with Applications*, 154:113–385, September 2020.

[2] Vitalik Buterin. Ethereum whitepaper, 2013.

[3] Kumar Ch and rakant. Implementing a Simple Blockchain in Java | Baeldung, September 2019.

[4] L Kadariya. Concurrency in Blockchain Based Smartpool with Transactional memory. 2018.

[5] J Meiners. Write your Own Proof-of-Work Blockchain.

[6] Z. Painter, P. K. Gayam, V. Cook, and D. Dechev. Parallel Hash-Mark-Set on the Ethereum Blockchain. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–5, May 2020.