Finding an Optimal Model for Cryptocurrency Mining

Danny Lu, Charlie Plachno, Syd Lynch
University of Oregon

Abstract - In this research paper we seek a model that would be optimal for mining cryptocurrency. We explore an alternative method that we refer to as the selfless model. The selfless model aims to maximize computing power over the entire network of machines instead of one machine looking to race against others for monetary gain. We test our model by implementing a Python script that mimics the proof-of-work calculation in a blockchain and measure the performance of various models. We compare the models as well as the performance of a single machine against multiple machines. Using data gathered from our tests and real world mining statistics, our findings show that it would take a significant amount of average machines in a network to match the processing power of a dedicated mining machine and perform competitively in a cryptocurrency with high hashing difficulty. We conclude that pooling resources is an optimal model of mining, but would not realistically be implemented with only average computing devices.

Keywords: cryptocurrency mining, proof-of-work, application-specific integrated circuit, blockchain

I. Introduction

Of the many emerging topics in modern technology and computer security, cryptocurrency is among the most prevalent and volatile. The topic is constantly in headlines with concerns of its rise and fall in value, questions of its stability, and an overall effort to determine the new currency's role in society. Since their inception with Bitcoin in 2009, decentralized digital currencies have exploded in popularity; Bitcoin itself boasting roughly 194,000 transactions per day [1] and an extremely large hashing difficulty.

Due to the overwhelming popularity of the largest cryptocurrencies, it would be an expensive and time-consuming endeavor to become involved in mining certain cryptocurrencies. Those most invested in the process are already well established with the hardware and knowledge necessary to remain competitive. The discouraging high barrier to entry is the inspiration behind our work. Is there an optimal model for mining cryptocurrency overall? Is there a method of mining competitive cryptocurrencies such as Bitcoin that is worthwhile for newcomers who lack highly efficient mining machines?

The goal of our research is to compare various methods of cryptocurrency mining with the intention of understanding their benefits and drawbacks, ultimately establishing a theoretical optimal model. We are particularly interested in the concept of dividing the intensive workload required for mining among multiple average computing devices. If such a model is feasible, then there may be more incentive for those uninterested in or incapable of buying expensive mining hardware to become involved in mining.

II. Background

A. Cryptocurrency

Cryptocurrency is an economic system that uses digital assets as a form of money. One of the properties of cryptocurrency is that the ledger that contains all transactions does not belong to any centralized entity. Instead, every actor in the cryptocurrency network has an identical copy of the ledger, making it very difficult for an attacker to fabricate transactions or create fake cryptocurrency to give to themselves [2].

B. Blockchain

This decentralized method of tracking currency is made possible by the blockchain model. A blockchain is a growing list of records that are represented by blocks. Every block is linked together in a linked list fashion. Each block

is secured by using a one way cryptographic hash. Every block also contains a hash of the previous block, a timestamp, and transaction data. Altering the components of a block would alter its hash thus affecting the rest of the blocks in the blockchain, because every block relies on the hash of the previous block to be linked.

C. *Proof-of-work*

Every block in the blockchain requires a proof-of-work calculation in order to validate the block. The proof-of-work calculation can be viewed as an intensive mathematical problem that requires a massive number of computations and great processing power in order to successfully solve.

The proof-of-work problem uses a base string that outputs a hash. We then have to find different variations of the base string and hash all those different variations to produce a hash that contains certain properties that we want. We can obtain different base string variations by simply adding a counter at the end of the base string, known as a nonce, that we can increment in order to produce an entirely different hash value. For example, Bitcoin uses the Hashcash proof-of-work system that tries to find a hash with a certain amount of zeros at the beginning of the hash.

III. Related Work

Given the popularity of cryptocurrency mining and its focus on hardware efficiency, there were many related works for us to research and refer to. We used these works in the following ways:

A. *Bitcoin mining acceleration and performance quantification*

This paper uses mathematical models and test results to find accurate numbers of hashrates coming from standard CPUs and GPUs and even finds hashrates resulting from simultaneous usage of CPUs and GPUs on the same machine, something that we had not previously considered. Their results were that no current CPU, GPU, or combination of both is powerful enough to compete with dedicated mining machines [3].

B. *On the security and performance of proof of work blockchains*

This paper has a very detailed overview of proof-of-work blockchain implementations and attempts to find security vulnerabilities in existing models and uses their findings to compare implementations based on trade-offs in security and performance. This overview is helpful for understanding certain aspects of the proof-of-work calculation and why it is necessary [4].

C. *Case study of the miner botnet*

This is a study of a specific botnet which utilizes the processing power of non-consenting victims by quietly infecting computers through malware and then running hashing scripts to contribute to a larger mining network. Our own model is essentially a fully consenting version of a botnet, sometimes referred to as a mining pool, so the failures and success of a botnet such as this one reflect on the ability of our own model [5].

IV. Design and Implementation

For our implementation, which we have named the *Selfless Model*, we have created a networked cryptocurrency mining simulation.

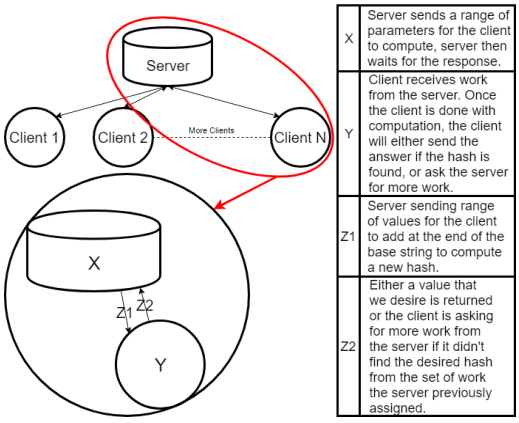

Figure 1: The server distributes all the work to each individual client and waits for the client to respond.

A server will send all connected clients a base string, a unique range of hashes to compute, and the target number of leading zeros desired in the hash. Once this information is received from the server, each client will compute the hash of the base string concatenated with the cryptographic nonce, iterating the nonce value until a suitable hash with the number of leading zeros necessary for the target is found or the upper bound of the

hash range is reached. The client will either send the server the nonce value that resulted in a suitable hash for that specific base string and target, or inform the server that the correct hash was not found. In addition, the client will relay to the server its current hashing performance in the form of hashes computed per second.

The communication between the server and clients described in our cryptocurrency mining simulation has multiple variations. The first two revolve around the server assigning each client an equally distributed, static value for the range of hashes to compute. In our implementation, what we consider a small range of hashes is between 1,000 - 100,000. The high range is then a number within 1,000,000 - 100,000,000. Having defined these ranges, our first model is a low range of statically distributed work, and our second model is a high range of statically distributed work.

Our third model is a dynamic distribution of work relative to each client's current performance. Initially, the server will assign the same amount of work to all clients as in the static models. However, each client will send the server the average number of hashes they are computing per second every time they finish computing their entire range of hashes. Using this performance information, the server will calculate how much work to assign each client based on their hashes per second. This system allows clients capable of computing a massive number of hashes per second to receive a proportionally large range of hashes to compute from the server, and for poorly performing clients, a smaller range of hashes. The dynamic model also ensures that we do not throttle a client's performance by lowering the amount of work if its performance dwindles.

We have also created a slightly modified version of our mining simulation that we are able to run on mobile devices using the application QPython3. The modified script is not connected to a network and instead calculates the proof-of-work of a base string on a single smartphone in a static fashion. This was implemented as a supplement to our project in order to have a basic understanding of the processing power of mobile devices. The idea behind this is to test the possibility of using smartphones to contribute to cryptocurrency mining as they are incredibly widespread and have less diverse hardware configurations.

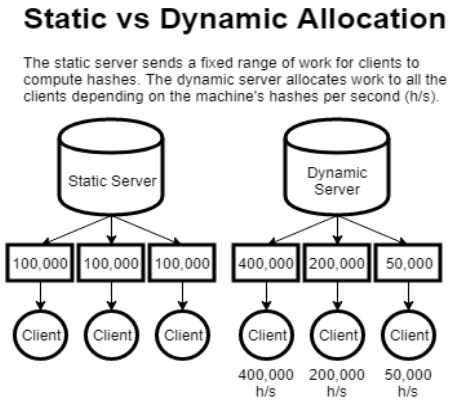

Figure 2: A demonstration of static and dynamic work distribution.

V. Results and Analysis

In testing our implementation we ran two different hashing functions, SHA256 and MD5, in a series of five 60 second tests. Our results conclude that MD5 performs much better than SHA256 in terms of calculating more hashes per second. We conclude that MD5 performs better due to its 128 bit hash size as opposed to SHA256's 256 bit hash.

When testing our network of laptops we used three different machines at varying CPU computing power. All three machines also run different operating systems: Mac OS X with an Intel i5-2415M processor, Windows 10 with an Intel i7-4500 and a UNIX-like desktop environment named XFCE running with an Intel Celeron 2955U. We ran three different models: static allocation with a small range of work, static allocation with a high range of work, and dynamic work allocation. We took the average of five tests for each model. Each model was applied to a network of computers as well as a single computer to measure the combined efforts of multiple machines compared to a single machine. We replicate this process with two different hashing algorithms thus producing sixty total tests. Less formally, we tested our hashing script modified for smartphones and compare the results to the performance of a single machine.

When allocating statically with small ranges of work our tests results show that the three

computers were able to output 622,825.15 h/s for SHA256 and 689,657.2 h/s for MD5. We used a range of 100,000 hash calculations allocated to each machine in the network to represent small payloads of work.

Statically allocating large ranges of work our results output 627,135.17 h/s for SHA256 and 708,257.78 h/s for MD5. We used a range of 1,000,000 hash calculations to represent large payloads of work.

Dynamically allocating ranges of work first sends a fixed size range of 1,000,000 to the clients. The clients then perform the hashing calculations and tells the server if it found the hash as well as notify the server on how fast it was able to calculate the hash. The server adjusts the next payload to the client based on their performance as described in our design. In the series of tests with the model of dynamic work distribution, our results show an average hashrate of 669,091.95 h/s for SHA256 and 704,498.7 h/s for MD5.

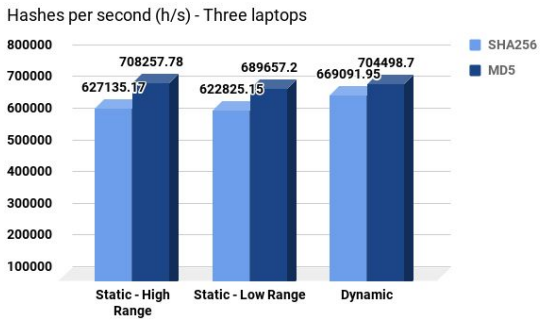

Figure 3: A comparison of hashing performance between two hashing algorithms and three work allocation models on three laptops working simultaneously.

The results of testing our simulated mining environment with three machines indicate a slight increase in hashing performance for the dynamic distribution model using SHA256. In our testing scenario, the most powerful of the three laptops is given the largest range of work which should allow for more hashes to be computed overall as the process is not completely throttled by the slower machines. The average hashrate of the dynamic model using MD5 is slightly lower than in the model of static distribution of high work ranges. While there are several reasons this could occur, such as the performance of the machines during the tests, we believe the dynamic model will more often result in a higher hash rate. The increase in performance using dynamic distribution should become clearer as the number of connected clients with varying hardware increases.

When testing the average hashrate in our simulated mining environment on a single laptop under the same conditions as the previous tests, our results show less of a performance increase for dynamically distributed work. There remains a clear advantage in speed when using the MD5 hashing algorithm over SHA256. The highest average hashrate for MD5 was 195,972.4 h/s using the model of statically distributing high ranges of work. For SHA256, the highest hashrate was 155,215.65 h/s using the dynamic distribution model. The lack of distinction between the static and dynamic models in this series of tests is unsurprising. The dynamic model is advantageous when the server has clients of varying power to choose from, granting more work to machines with higher processing power. With only one machine, the model is approximately static.

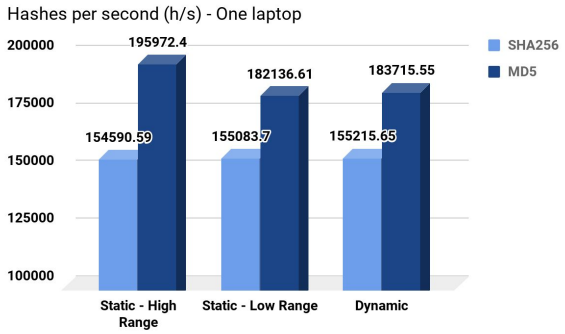

Figure 4: A comparison of hashing performance between two hashing algorithms and three work allocation models on a single laptop.

These results, while not unexpected, accentuate the value of splitting the work among multiple machines. Within the same one minute timeframe, a single laptop will compute approximately 0.30 times the amount of average hashes per second of three laptops working simultaneously.

To compare the productivity of this model to an ASIC S9 Antminer, it would take approximately 91 million laptops clocked at our measured 154,590 h/s to match the Antminer's

14TH/s (14 trillion h/s). A top competitor in Bitcoin has a facility of around 25,000 S9 Antminers which means that they have a collective hashing rate of 350,000TH/s [7]. It would take approximately 2 trillion of our laptops to match this.

The results of using our script modified for smartphones on a Google Pixel with a Qualcomm Snapdragon 821 processor are surprising. The average hashrate using a low range (100,000) of hashes is 91,666 h/s for SHA256 and 101,666.65 h/s for MD5. Using a high range (1,000,000) of hashes, the averages are 83333.31 h/s for SHA256 and 99999.98 h/s for MD5. This implies that the average hashrate on a high-end mobile device is only around 0.5 times the hashrate of a single laptop. Running this script for our series of test caused a noticeable drain on the battery.

VI. Complications

Many difficulties were encountered throughout the course of our project. One of the largest obstacles was determining an effective method of dynamically distributing a range of hashes to each client relative to their hashing performance. An initial idea was to make use of a database of hardware and have each client report their specific model of CPU to the server so that they could be ranked by processing power. While potentially effective, this method would be beyond the scope of the problem and project. The most logical measure of a client's performance is their hashrate. Our implementation maintains the amount of time it takes for each client to compute their assigned range of hashes, translating to hashes per seconds. The server divides the client hashrate by a base amount and sends the client a range of hashes determined by product of the ratio and the base amount.

Generating data that can be related to the performance of actual cryptocurrency mining was another hindrance. The implementation of our selfless model does not take advantage of a dedicated graphics card, instead only utilizing a small portion of CPU power. This is partly by design, as we envisioned this network of shared mining still allowing machines to be usable for other tasks without a negative effect on performance. However, profitably mining the larger cryptocurrencies is impracticable with only CPUs and largely ineffective without high-end GPUs. There is a clear performance advantage in cryptocurrency mining with a GPU due its abundance of cores which allow it to parallelize well and perform complicated mathematical computations with ease [6]. While we would have liked to test the performance of our simulated mining environment on a GPU, implementing this is another task that would demand more time and knowledge than available due to the restrictions of Python. This restraint made comparing our data to the observed performance of actual mining difficult.

The topic for our project caused issues with maintaining a clearly defined goal. We initially set out to establish a theoretical optimal model specifically for Bitcoin mining, determining the fastest and most efficient method. This was then generalized to all cryptocurrency mining as we became more aware of the number of digital currencies available. In addition, we wanted to explore the concept of pooling processing power to increase mining capabilities and did so through our simulated mining implementation. However, this seemed to occasionally become the central focus of the project, causing our research to conflict with our implementation.

VII. Conclusions

After collecting data from our tests and measuring our model's performance we conclude that, with our measured hashrates, networking average computers can outperform a dedicated cryptocurrency mining machine. However, it would take so many that splitting up the rewarded cryptocurrency would result in minimal monetary gain for every constituent in the network. The hashrate that we calculated for this test is realistically a lower range estimate for the average laptop but our conclusions remain unchanged. Realistically, people would not offer up their computing power for such minor gain and it would take a huge number of people to get involved for a reward to be likely at all.

Cryptocurrency mining operates as a race between all contributing agents to find the hash and to reap the reward. Very popular cryptocurrencies such as Bitcoin and Ethereum have contributors with so much processing power that our model would have very little chance of finding the correct hash before someone else. However, our selfless model could potentially still do well in smaller cryptocurrencies since the proof-of-work calculations are less complicated and more feasibly solvable with a network of average computing machines.

One of the largest considerations we had when figuring out a model was whether or not that model would make money, but another application of this method could be to give money instead of making it. Popular communities like Twitch, Patreon, and YouTube allow consumers to donate to their favorite content creators through watching advertisements or directly donating funds. However, networking a number of consumers to all pitch in their processing power instead of donating actual funds could be a reality through this process. An example similar to this concept can be found on the popular online news website Salon, which offers visitors of their website an option to view advertisements or to provide some of their computer's processing power for cryptocurrency mining, a method that has received a noticeable amount of criticism [8]. While our method only provides a very small gain per individual, a service that would reap the rewards of all the machines contributing to it could actually find the method very worthwhile. Although this idea has already been conceived or implemented in some ways, hopefully our data and test results can provide concrete evidence that using a network of average machines can still be used in a variety of applications.

VIII. Lessons Learned

One of the biggest realizations for us as we went through this research was that the computing speeds of our everyday computers were so much smaller in comparison to dedicated mining machines. We all assumed that with very optimistic approximations it would be fairly feasible to collect enough machines on a network to compete in even the biggest cryptocurrencies. Our hypothesis on this was fairly off base as we realized just how much computational power people could amass with large mining pools of dedicated machines. More specifically, we had no concept of the sheer amount of power and efficiency Application Specific Integrated Circuit (ASIC) machines are capable of. While the Antminer S9 is considered one of the most efficient ASIC machines, a more affordable option is the AvalonMiner 761, costing $1,860 and advertising 8.8 TH/s [9]. This is around the price of many modern laptops and has the ability to compute monumentally more hashes than even a huge network of laptops could achieve.

While not specifically related to our conclusions and results, we learned a great deal when designing our simulated mining environment. It was necessary to understand how to create a network in Python and maintain communication between a server and clients. Sending the necessary information in each packet to and from the server proved difficult and we learned how to correctly parse packet information for our purpose. We also now understand more about firewalls and the necessary steps of allowing our application through a firewall without completely disabling its security features.

IX. References

[1] "Bitcoin Charts & Graphs - Blockchain," *Blockchain.info*. [Online]. Available: https://blockchain.info/charts.

[2] S. Dziembowski and Stefan, "Introduction to Cryptocurrencies," in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15, 2015, pp. 1700–1701.

[3] J. A. Dev, "Bitcoin mining acceleration and performance quantification," in 2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE), 2014, pp. 1–6.

[4] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the Security and Performance of Proof of Work Blockchains," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16, 2016, pp. 3–16.

[5] C. Czosseck, R. Ottis, K. Ziolkowski, Institute of Electrical and Electronics Engineers., and NATO Cooperative Cyber Defence Centre of Excellence., 2012 4th International Conference on Cyber Conflict : proceedings. IEEE, 2012.

[6] S. Che, J. Meng, J. W. Sheaffer, and K. Skadron, “A Performance Study of General Purpose Applications on Graphics Processors.”

[7] “A Tour of One of the Largest Bitcoin Mining Operation,” *Altcoin Today*, 21-Aug-2017. [Online]. Available: https://altcointoday.com/tour-of-one-of-the-largest-bitcoin-mining-operation/.

[8] J. Morse, “Salon's new business plan is running cryptocurrency-mining malware on your computer,” *Mashable*, 13-Feb-2018. [Online]. Available: https://mashable.com/2018/02/13/salon-coinhive-cryptocurrency-monero/#hftn.XWzpOqG.

[9] “Mining hardware comparison,” Mining hardware comparison - Bitcoin Wiki. [Online]. Available: https://en.bitcoin.it/wiki/Mining_hardware_comparison.

X. Appendix

A. *hashing.py*

Hashing.py can be viewed as our client. This file only contains four functions: proof-of-work, check_hash, decodeMess, and main. In main, the client is prompted to type in the public IP address of the server to connect to. Once connected the client receives a range of hashes to compute from the server. The message is decoded using the decodeMess function which essentially is just a string parser to get all of the variables we need to pass into the proof-of-work function (Variables are: start, end, target, base_string). Proof-of-work calculates the hashes by adding a nonce at the end of the base string and calls the check_hash function to see if there is zeros at the beginning of the hash that equals to the number of zeros we want specified by target which was decoded when decodeMess was called.

B. *server.py*

Server.py has only two functions: send_work and main. In main, the server binds to a port and allows clients to connect to the server using the socket library. Every time a client connects to the server, the server appends it's IP address into a dictionary where it stores the IP address of all the connections as well as the hashes per second associated with each IP address. The server creates a thread using Python's threading library for every client that connects and calls send work which looks are the global counter and allocates a certain range to be send to the client to computer. The created thread is in charge of listening to the client and waiting until it's done with the work range that the server sent. When it hears back from the client, if the client didn't find the hash that meets our criteria then the server sends more work to the client. This process repeats until the desired hash is found. There is a global count that the server keeps track of to make sure no clients are computing the same work. We implemented a condition variable to lock the global counter to make sure there isn't any race conditions when allocating work.

C. *server_dynamic.py*

Server_dynamic does exactly what server.py does with the exception of how work ranges are allocated. Server_dynamic listens for the client, when the client responds it also sends the server how many hashes per second it is capable of computing. The server takes that information and applies an algorithm to determine how much work to allocate to the client based on their hashes per second. This all happens in real time so if a client's hashes per second is throttling, we can adjust the number of work to give to make sure we are not overwhelming the client in case it's running other operations that is taking up it's computing power.

D. *hash_mobile.py*

This file is a modified version of hashing.py, designed to work as a standalone script without any network capabilities. It possesses the same functions as hashing.py, but allowed us to test the hashing performance of a smartphone.