Randomized Approach to Finding the Optimal Blockchain Miner Configurations for Maximum Energy Efficiency

Sharan Duggirala and Siddharth Kulkarni[1]

Abstract— **Blockchain is a decentralized transaction and data management technology developed primarily for the Bitcoin cryptocurrency. Applying blockchain to the cloud efficiently, which will allow on-demand, secure and low cost access to computing infrastructures, may not be possible currently. This is due to the limited computing capacities, to run distributed applications, and the inefficient power consumption models pertaining to Blockchain technologies. We believe that exhaustive research on the variance of different parameters such as different hash functions and CPU threading to achieve the most energy efficient configurations for blockchain based mining, has not been conducted yet. Therefore by using a simulated blockchain network have analyzed the energy efficiency of miners against various parameters to determine optimal miner configurations for the most energy efficient blockchain mining.**

I. TABLE OF CONTENTS

II. LIST OF ACRONYMS

1) **BTC** - Bitcoin
2) **HPC** - High Performance Computing
3) **PoW** - Proof of Work
4) **P2P** - Peer to Peer
5) **CPU** - Central Processing Unit
6) **GPU** - Graphical Processing Unit
7) **ASIC** - Application-Specific Integrated Circuit
8) **SPI** - Serial Peripheral Interface Bus
9) **USD** - United States Dollar
10) **FIFO** - First in First out
11) **PCR** - Pipe Control Register
12) **SHA** - Secure Hash Algorithm
13) **RAM** - Random access memory
14) **J/GH** - Joules per GigaHash
15) **PH/s** - Peta Hashes per second
16) **EVM** - Ethereal Virtual Machines

III. INTRODUCTION

Blockchain is a **decentralized** transaction and data management technology developed first primarily for the cryptocurrency Bitcoin. The interest in Blockchain technology has been steadily increasing, since the idea was coined in 2008[2]. It essentially is a continuously growing list of records, called **blocks**, which are linked and secured using cryptography. Each block typically contains a hash pointer as a link to a previous block, a **timestamp** and **transaction data** (see *Fig 2* for more details). By design, blockchains are inherently resistant to modification of the data.

A blockchain can serve as "an open, **distributed ledger** that can record transactions between two parties efficiently and in a verifiable and permanent way[2]. For use as a distributed ledger, a blockchain is typically managed by a **peer-to-peer** network, collectively adhering to a protocol, for validating new blocks. Once recorded, the data in any given block cannot be altered retroactively without the alteration of all subsequent blocks, which needs a collusion of the network majority.

Blockchains are secure by design and are an example of a distributed computing system with high **Byzantine fault tolerance**[3]. Due to the nature of a decentralized consensus, blockchains are potentially suitable for the recording of events, medical records, and other record management activities. The information about every transaction, completed in Blockchain, is shared and available to all nodes. This attribute makes the system more transparent, than a system which employs centralized transactions involving a third party. In addition, the nodes in Blockchain are all anonymous, which makes it more secure for other nodes to confirm the transactions. Bitcoin was the first application that introduced Blockchain technology. Bitcoin created a decentralized environment for cryptocurrency, where the participants can buy and exchange goods with digital money.

A popular question, *Is Blockchain needed for building a Distributed Cloud?* needs to be addressed. There is a growing demand for computing power from scientific communities and industries to run large applications and process huge volumes of data. The computing power to run **Big Data** application is most often provided by **HPC** and Cloud infrastructures. A Blockchain- based Distributed Cloud will allow on-demand, secure and low-cost access to the most competitive computing infrastructures.

However, some research needs to be conducted within the blockchain environment to ameliorate the disadvantages that are hampering its widespread usage. The fact that blockchains offer very limited computing capacities, to run distributed applications: a few *kb* of storage, very inefficient virtual machines and a very high latency protocol, are its major disadvantages. Eventually blockchain technologies will evolve to overcome some of these issues. We plan to research

[1]S. Duggirala & Siddharth Kulkarni are with the Department of Computer Science, San Jose State University, 1 Washington Square, San Jose, CA, USA

and tackle one of the primary issues with **Blockchain** and **Proof of Work**, which is its inefficient energy consumption.

IV. Literature Survey

A. Exploring Miner Evolution in Bitcoin Network

In order to better understand how to optimize the Blockchain, the most popular usage of blockchain and relevant parameters, need to be examined. As of now, blockchain is primarily used in the **Bitcoin** Network which was conceived to support the popular Bitcoin currency[1]. Through Wang, Liu[5] we understand that a user of the Bitcoin network can create bitcoins by packing and verifying new transactions in the network using their own computational power.

With a given difficulty value of D, if N blocks are mined in a day, the aggregate hash rate per day of the entire network is given in below[5]:

$$H_{\text{total}} = \frac{N \times D \times 2^{32}}{86,400}$$

If the hardware of hash rate H, based on equation detailing the expected time for a miner to generate a valid block, works twenty four hours a day, the expected amount of *BTC* mined daily is given below:

$$N(t,h) = \frac{H \times 86,400}{D(t) \times 2^{32}} R$$

Where $D(t)$ is the difficulty value in day t and R is the number of bitcoins being awarded for each block. If the hardware power consumption is PKW, and the price is $\eta(t)$ per KWh, the daily electricity bill is $24P\eta$. If we only consider the electricity operational cost, the profit rate $r(t)$, can be represented as below:

$$r(t,H,P) = N(t,H)\rho(t) - 24P\eta(t)$$

To maintain a profit the miners need for the *computation-over-power* efficiency to satisfy:

$$\frac{H}{P} > K\frac{\eta(t)D(t)}{R\rho(t)}$$

Where K is a constant in above[5]. Wang and Liu also describe how the miners are paying a zero-sum computation race game. Since miners are constantly motivated to increase their computational power, and, the bitcoin price is kept flat, as to keep a steady number of coins that can be mined each day. This race will end when no existing machine can satisfy the computation-over-power efficiency equation above, thus concluding our findings from this particular paper.

B. Towards a More Democratic Mining in Bitcoins

Bitcoin has enjoyed superiority compared to other cryptocurrencies, but, has also attracted attackers to take advantage of possible operational insecurities that plague its system.

The authors have cited various techniques to solve these insecurities. The first being the paper titled *Preventing Selfish Mining*[10]. **Block races** consists of mining pools pursuing selfish mining to prevent the blocks of other users from being added into the blockchain. This can lead to 51% attack. The authors suggest a fair mining process and thus remove the chances of block races and selfish mining. The second technique is the titled *Guaranteed Fixed Generation Rate*[11]. The bitcoins are generated at a highly variable output at an average of 10 minutes. This variation can lead to problems like transaction malleability, which in turn can lead to frauds.

The authors propose to modify the target achieving proof-of-work protocol through **minimum hash generation** by miner nodes across the Bitcoin network. The user with the minimum hash after a given amount of time (ten minutes was primarily decided upon on this paper) gets the mining rights.

The changes introduced, are modifications to the present *BTC* header. The target field has been replaced by the *BTC* address of the miner. The advantages of democratic mining are *BTC* generation at a **fixed rate**, which can be adjusted. Decentralization of the *BTC* mining process leads to a reduced power consumption (**20%** of the original power consumption).

To conclude, the authors of this paper address the problem of 51% attack and introduce a new defense against this problem. Modification of the present block header by introducing some extra bytes and utilizing the timestamp more effectively in the hash generation, suggests an alternative to the existing Proof-of-Work scheme. The proposed approach not only relies on finding a hash values lower than the target, but awards the miner involved in generating the minimum hash value across the entire distributed network. Fraudulent activities are easier to determine due to effective use of the Timestamp. The new scheme thus introduces fair competition among the miners. Moreover, it facilitates the generation of *BTC* at a fixed rate hence developing a new scheme of an energy-efficient *BTC* network.

C. Bitcoin Mining Acceleration & Performance Quantification

We can learn more about the mining through Jega Anish Devs paper[12]. The work expended in generating a **Proof-of-Work** can be considered as the computation cycles involved in the repeated *SHA-256* computation needed to discovering the required hash seed. The process for any random jth step, in terms of both GPU and CPU mining is shown in Figure 5.Dev proposes a novel technique where the computational prowess of both the CPU and the GPU are used in the mining process. In order to achieve this, separate algorithms were described for each hardware.

The following steps were performed on an *Nvidia* graphics card that uses the *CUDA* library:

1) **Allocation** of resources on the device, such as a single instance of the structure holding block header details that is used in hashing and a copy of the best nonce for each thread being executed in the device.

2) **Initialization** of the variables in the kernel function for concurrent SHA256 operations.
3) **Execute** the SHA256 computation on the trial block structure which is filled with static data that was received during the usage of the BTC protocol and with an incremented trial.
4) **Test** if the hash obtained from *step 2* is smaller than the target hash given by the difficulty.
5) If success, **report** the winning nonce. Otherwise increment the nonce and perform *step 2*.

The following algorithm was described for the mining process on a CPU:

- **Allocation** of the resources as described before, but on the RAM.
- **Initialization** of the variables to concurrently provide an id for starting values of the nonce to each parallel instance of the SHA256 operation.
- **Execute** SHA256 operation in the same manner as described in GPU mining.
- **Test** hash and on success, report winning nonce. Otherwise, increment the nonce and perform *step 2*.

The approach discussed in Devs paper is not expected to be comparable with custom hardware based mining, but provide a relevant boost to hash rates for non custom equipment users.

D. CoinTerras First-Generation Cryptocurrency Mining Processor for Bitcoin

This paper describes the architecture and implementation of **CoinTerras** first generation *BTC* mining processor, **GoldStrike-1** and how it was used to design a complete *BTC* mining machine called **TerraMiner-IV**. Because of high power density in the *BTC* mining processor, delivering power and cooling to the die posed enormous challenges. This paper describes some of the solutions adopted to overcome these challenges.

As the difficulty of *BTC* mining continued to increase, it had spawned a new design orientated industry that develops powerful custom computing hardware for *BTC* mining applications. From the design and production of specialized integrated circuits for *BTC* mining processors, this new industry provides a complete solution by integrating these processors into custom *BTC* mining appliances that can easily be used by individual consumers as well as large datacenters. A defining attribute of *BTC* mining hardware, is that, it requires to be operated at peak performance consistently at all times of the day. In contrast, a general-purpose computing processor or a graphics processor uses only a small fraction of the computing resources at peak level at any given instant of time. This particular usage model for *BTC* mining hardware results in extremely high and sustained power consumption, creating new design challenges. A couple of design challenges to overcome includes, maximizing the energy efficiency of the application-specific integrated circuit (ASIC) processor and achieving efficient and stable power transmission and thermal solutions, while keeping the cost structure viable for *BTC* mining operations. This article describes the architecture and implementation of CoinTerras first-generation *BTC* mining processor, GoldStrike 1, and how these processors are integrated into a complete *BTC* mining appliance called Terra-miner IV.

The authors of this paper detail some future trends, such as, moving the Bitcoin mining operations to data centers locations with the lowest electricity and cooling cast costs. There has also been a lot of focus on increasing the ASICs energy efficiency, by employing more advanced process nodes, lower supply voltages, and a highly configurable ASIC back-end design. To increase the energy efficiency, the switching capacitance along with the supply voltage must be minimized. Current trends point towards older process technologies for lower power and lower hash rate chip design on a cheaper process node, and deploy a massive amount of these chips to get a higher total hash rate.

Based on the exchange rate and the operational costs, at the time of the conception of the paper, Cointerra is able to perform with an efficiency of $0.8J/GH$(Joules/GigaHash) at full clock, with a Hashrate of $179.7PH/s$(PetaHashes/second).

In conclusion this paper discusses the architecture and implementation of CoinTerras first generation bitcoin mining processor. It also explains various techniques to reduce power consumption and still have efficient operational capacity.

For the purpose of our research we built our own private blockchain using python and python based framework ”flask”. A simulation was conducted using randomized miner configurations over a period of time. We recorded the energy efficiency values for each miner and conducted further analysis on relation between different miner configurations and energy efficiency.

V. Existing Industry Technology

a) VISA: Visa is an American multinational financial serves corporation headquartered in Foster City, California, United States. It facilitates electronic funds transfers throughout the world, most commonly through VISA branded credit cards and debit cards. In Oct, 2016, VISA announced their partnership with blockchain enterprise company Chain in which the two firms decided to develop a simple, fast and secure way to process B2B payments globally. The Visa B2B Connect platforms pilot is expected to launch in 2017, thus indicating a connection between the USPTO digital asset network patent and the new B2B solution. [15]

b) Ethereum: Although commonly associated with Bitcoin, blockchain technology has many other applications that go way beyond digital currencies. In reality, Bitcoin is only one of several hundred applications that use blockchain technology today.

Until recently, building an application that uses the blockchain technology has required a complex background in coding, cryptography, mathematics as well as significant resources. Ethereum is an open distributed public blockchain network that facilitates easier development of these applications. While blockchain in the context of Bitcoin is used to track ownership of digital currency(*BTC*), Ethereum focuses on running the programming code of any decentralized application[16].

In the Ethereum, instead of mining for bitcoin, miners work to earn Ether, a type of crypto token that fuels the network. Beyond a tradeable cryptocurrency, Ether is also used by application developers to pay for transaction fees and services on the Ethereum network.

Before the creation of Ethereum, blockchain application were designed to do a very limited set of operations. With the invention of the **Ethereal Virtual Machine**(EVM), a Turing complete software that runs on the Ethereum network, anyone can run a program, regardless of programming language, on Ethereum. Instead of having to build an entirely original blockchain for each new application, Ethereum enables the development of countless applications on the same environment. For example, Dapp provides its users with a P2P electronic cash system that enables online Bitcoin payments. Due to the nature of Ethereum, a particular centralized service can be adapted to be more decentralized.

VI. Implementation

A. Server

The server program is designed to process requests and deliver data to users and miners over the entire block chain network. The functionality of the server can best be described by the various routes through which requests are made and data is sent and received.

a) Transactions Manager: This route is called whenever a new transaction has been made between two users. This transaction is then added to a queue of unverified transactions, which waits till it is verified by miners.

Listing 1: Transaction Route

```
@node.route('/txion', methods=['POST'])
def transaction():
  # On each new POST request,
  # we extract the transaction data
  new_txion = request.get_json()
  # Then we add the transaction to our list
  this_nodes_transactions.append(new_txion)
  # Because the transaction was successfully
  # submitted, we log it to our console
  print "New transaction"
  print "FROM: {}".format(new_txion['from']
  .encode('ascii','replace'))
  print "TO: {}".format(new_txion['to']
  .encode('ascii','replace'))
  print "AMOUNT:
      {}\n".format(new_txion['amount'])
  # Then we let the client know it worked
      out
  return "Transaction submission
      successful\n"
```

b) mine - Mining Hash Verification: This route is called when a miner sends an answer for verification. On successful verification, a new global challenge is set and the miner is awarded with a coin.

A new challenge is set according to the difficulty level, the computation of which is described later on within this report.

Listing 2: Mine Route

```
@node.route('/mine', methods=['POST'])
def mine():
  # Get the miner information including
      the answer
  informationMiner = request.get_json()
  miner_information_dict
  [informationMiner['miner_address']] =
      informationMiner
  answer = informationMiner['answer']

  # Check if the Challenge and Answer are
      truly equal
  if(len(this_nodes_transactions) != 0 and
      answer == challenge):

  # Create a new global challenge, since
      challenge has already been solved
  global challenge
  [challenge, d_level] =
      refresh_challenge(miner_information_dict)
  print str(challenge) + " new challenge"

  # Let the client know we mined a block
  return "\n" + json.dumps({
   "index": new_block_index,
    "timestamp": str(new_block_timestamp),
    "data": new_block_data,
    "hash": last_block_hash
   }) + "\n"

  # In the case where there is nothing to
      mine (no transactions)
  else:
    return "Try Again"
```

c) info - Get current Hash challenge: This route is used to receive information about the current challenge. Also new miners are added to the list of miners that resides on the server. This serves the purpose of deciding future challenges.

Listing 3: Information Route

```
# Get Challenge
@node.route('/info', methods=['POST'])
def get_info():
 informationMiner = request.get_json()
 miner_information_dict
  [informationMiner['miner_address']] =
     informationMiner
 return challenge
```

d) board - Get coins earned by each miner: This route returns the data of coins earned by each miner, up until the time the request was made.

Listing 4: Board Route

```
@node.route('/board', methods=['POST'])
```

```
def board():
 return json.dumps(miner_information_dict)
```

e) blocks - Internal function call: This route is an internal function call, that returns the *blocklist* i.e all blocks connected to the blockchain to the network.

Listing 5: Blocks Route

```
# Internal Blocks Method
@node.route('/blocks', methods=['GET'])
def get_blocks():
  chain_to_send = blockchain
  for i in range(len(chain_to_send)):
    block = chain_to_send[i]
    block_index = str(block.index)
    block_timestamp = str(block.timestamp)
    block_data = str(block.data)
    block_hash = block.hash
    assembled = json.dumps({
    "index": block_index,
    "timestamp": block_timestamp,
    "data": block_data,
    "hash": block_hash
    })
    if blocklist == "":
      blocklist = assembled
    else:
      blocklist = blocklist + assembled
  return blocklist
```

f) print_blocks: This is a debugging route that is used for returning and displaying information of all the blocks currently on the blockchain.

Listing 6: Blocks Route

```
#External Blocks Method
@node.route('/print_block', methods=['GET'])
def print_blocks():
  #print '\nLength
     ',len(this_nodes_transactions)
  chain_to_send = blockchain
  blocklist = ""
  for i in range(len(chain_to_send)):
    block = chain_to_send[i]
    block_index = str(block.index)
    block_timestamp = str(block.timestamp)
    block_data = str(block.data)
    block_hash = block.hash
    assembled = json.dumps({
    "index": block_index,
    "timestamp": block_timestamp,
    "data": block_data,
    "hash": block_hash}, sort_keys=False,
        indent=4)
    if blocklist == "":
      blocklist = assembled
    else:
      blocklist = blocklist + assembled
  return blocklist
```

B. Miner

The Miner is a stand-alone program of the `F.Block`. The Miner system within the `F.Block` system can be divided into five different different functions:

a) Define Miner Class Objects: The Miner Class defines each miner with the following characteristics:

- *Address*
- *CPU*
- *GPU*
- *Coins/Currency Earned*

Listing 7: Miner Class with Constructor

```
class Miner:
    def __init__(self, miner_name,
         miner_cpu, miner_gpu):
        self.miner_address = miner_name
        self.miner_coins_earned = 0
        self.miner_cpu = miner_cpu
        self.miner_gpu = miner_gpu
```

b) Detect Miner CPU and GPU configurations: The Miner CPU and GPU configurations are detected using following two steps:

- OS Detection: The `hashattr()` function provided by python to detect the OS Miner is running on.
- CPU/GPU Detection: This is described below.

Listing 8: Detection for Linux, Unix OS

```
ncpus = os.sysconf("SC_NPROCESSORS_ONLN")
```

Listing 9: Detection for Mac OS

```
os.popen2("sysctl -n hw.ncpu")[1].read()
```

Listing 10: Detection for Windows OS

```
os.environ["NUMBER_OF_PROCESSORS"]
```

c) Get Challenge from Server: Once Miner has been initialized, it sends a request to server to return the current hash to solve. In this process it also informs the server of its existence and its configurations. This information is used by the server to keep count of number of miners and decide difficulty level.

Listing 11: Get Challenge from Server

```
req = urllib2.Request(
 "http://localhost:5000/info",
 json.dumps({ 'miner_address' :
     miner.miner_address, 'coins_earned' :
     miner.miner_coins_earned ,
     'miner_cpu' : miner.miner_cpu,
     'miner_gpu' : miner.miner_gpu}),
     headers={'Content-type':
     'application/json', 'Accept':
     'application/json'})
```

```
    response = urllib2.urlopen(req)
    challenge = response.read()
```

d) Solve Hash Challenge provided by server: Once a challenge has been received from the Server, the $solveHash()$ method is called with the received hash sent as a parameter.

Listing 12: solveHash() method

```
def solveHash(challenge):
    # Now we can attempt to find the solution
    ping_index = 0
    found = False
    while found == False :
        ping_index += 1
        answer = ''.join(random.choice(
        string.ascii_lowercase +
             string.ascii_uppercase +
             string.digits)for _ in range(4))
        if ping_index == 1000000 :
            ping_index = 0
            req2 = urllib2.Request(
            "http://localhost:5000/info",
            json.dumps({ 'miner_address' :
                 miner.miner_address,
                 'coins_earned' :
                 miner.miner_coins_earned ,
                 'miner_cpu' : miner.miner_cpu,
                 'miner_gpu' :
                 miner.miner_gpu}),
                 headers={'Content-type':
                 'application/json', 'Accept':
                 'application/json'})
            response2 = urllib2.urlopen(req2)
            challenge = response2.read()
            print challenge
        if answer == challenge :
            found = True
            #print str(miner.miner_address) + "
                 has mined a coin"
            return answer
```

e) Send Answer to Server: When the Miner has found a solution, it sends its answer to the server for verification.

Listing 13: Get Challenge from Server

```
   my_answer = solveHash(challenge)
    req3 = urllib2.Request(
        "http://localhost:5000/mine",
        json.dumps({ 'miner_address' :
            miner.miner_address,
            'coins_earned' :
            miner.miner_coins_earned ,
            'miner_cpu' : miner.miner_cpu,
            'miner_gpu' : miner.miner_gpu ,
            'answer' : my_answer}),
            headers={'Content-type':
            'application/json', 'Accept':
            'application/json'})
   response3 = urllib2.urlopen(req3)
   the_page3 = response3.read()
```

C. Proof of Work

The Proof of Work system is one of the most essential aspects of a cryptocurrency. The **PoW** system within `F.Block` can be divided into three different different functions:

a) Challenge Initialization: This is called from the server before any of the other computations are performed. The responsibility of this function is to discover the total computation power of the network at the very beginning (which could be a single miner being service by the server) and creating the initial challenge. The **CPU** and **GPU** values are properties of the dictionary `miner_information_dict` and therefore can be retrieved when the dictionary is passed as an argument from the server. The code is shown below:

Listing 14: Discovering the Total Computational Power

```
  for key in peer_nodes:
     tot_gpu += peer_nodes[key].miner_gpu
     tot_cpu += peer_nodes[key].miner_cpu
  if (tot_gpu + tot_cpu > known_gpu +
       known_cpu):
     known_gpu = tot_gpu
     known_cpu = tot_cpu
```

In the above given code, `peer_nodes` is the `miner_information_dict`, and we sum up the computational power of every miner. Since `tot_gpu` and `known_gpu` are initialized to 0, the system is sure to recognize the new computational power of the combined network. It is essential that we record the initial computational power of the system, since it is required in the computation of the difficulty level later on.

We can now create the challenge for the miners to solve.

Listing 15: Creating the Challenge

```
 answer = ''.join(random.choice(
        string.ascii_lowercase +
        string.ascii_uppercase +
        string.digits)
        for _ in range(4-difficulty_level))
```

As it can be seen above, the challenge is a combination of uppercase and lowercase alphabets along with digits. This gives the miners a very robust challenge to solve, when it gets hashed by the server using **SHA-256**.

b) Refreshing the Challenge: Once the challenge has been initialized and solved, a new challenge needs to be created every time a challenged is solved, which is the case, when a new block has been mined. The first response of this method is to determine the difficulty level of the system based on its total computational power. This is further described in the forthcoming section. Once the `diff_level` has been determined, we use it to pad our challenge with zeros using:

Listing 16: The Process of Padding

```
for i in range(difficulty_level):
    challenge+='0'
```

Every zero that is added to the start of the challenge doubles(amortized) the difficulty of the challenge. It must be noted that we are not processing the challenge through a SHA-256 algorithm as of now, since this would require CPU and GPU capabilities that are beyond the scope of a simulation. Once we have padded the challenge, we can randomly select the remaining characters once again.

Listing 17: Creating the Challenge 2

```
answer = ''.join(random.choice(
            string.ascii_lowercase +
            string.ascii_uppercase +
            string.digits)
            for _ in
                range(4-difficulty_level))
```

c) Difficulty Level Adaption: In order to determine the difficulty level, the total computational power of the system is calculated once again. There are four different situations that may occur here.

- There may be a total of zero miners within the system. In this case there does not need to be a change to the difficulty level.
- The C_{total} (Total Computational Power within the system) may have remained the same as the previous recorded computational power. In this case, there does not need to be a change to the difficulty level.
- The C_{total} may have doubled. In this case, we increase the difficulty level by 1, along with setting a new total GPU and CPU.

Listing 18: Increasing the Difficulty Level

```
if (tot_gpu + tot_cpu > 2*(known_gpu +
    known_cpu)):
    difficulty_level+=1
    known_gpu = tot_gpu
    known_cpu = tot_cpu
```

- The C_{total} may have halved. In this case, we decrease the difficulty level by 1, along with setting a new total GPU and CPU.

Listing 19: Decreasing the Difficulty Level

```
if (tot_gpu + tot_cpu < 2*(known_gpu +
    known_cpu)):
    difficulty_level-=1
    known_gpu = tot_gpu
    known_cpu = tot_cpu
```

D. Transaction Model

At the heart of `F.Block` lies a probabilistic model that powers the transactions that the server receives over time. The transaction model within `F.Block` can be divided into three different different functions:

a) Refreshing Transactions: This is the only function that is available to the server. It ties the rest of the program together as it calls on the other three segments of this model. It first gathers the two transactors involved within the transaction and then proceeds to find a random amount of money that has been transacted between them.

Listing 20: Random Transaction Amounts

```
transaction_amount = random.randint(1,101)
```

It proceeds to deduct the transaction amount from the transactors involved, keeping the amounts in check and never exceeding the amount that the sender has. This information is finally relayed back to the server.

b) Creating Transactors: This model holds the information of each and every transactor through the class:

Listing 21: The Transactor Class

```
class Sample_transactor_data:
  def __init__(self, miner_address, coins):
    self.miner_address = miner_address
    self.coins = coins
```

A random address and a random amount of money were given to each transactor. For the purpose and the scale of the experiment, we have chosen to create a **100** different transactors.

c) Probabilistic Model: The most interesting segment and function within the transaction's probability model is the model itself. There are two different scenarios in which this function can be called. The first one is when the server is being initialized. In this case the transactors need to be initialized too with the function described above. In order to fully realize the probabilistic model, the data has been abstracted to the `Card` class:

Listing 22: The Card Class

```
class Card:
  def __init__(self, miner_hash, prob):
    self.miner_hash = miner_hash
    self.prob = prob
```

When initializing the transactors data, such as the mining address and the probability of the Card(transaction) being chosen, are relayed onto the function `deck_chooser`, which is described in the next section. In order to the calculate the probability of each card being chosen, a simple equation has been used:

$$P_{\text{i}} = \frac{C_{\text{i}}}{C_{\text{total}}}$$

Where P_{i} is the probability of a particular transactor getting chosen, while C_{i} is the number of coins that a transactor currently possesses, and C_{total} is the total number of coins that are being held by all the different transactors.

Listing 23: Code to Calulate Probabilities

```
# First Calculate the Total Number of Coins
for i in range(len(transactor_data)):
 total_coins += transactor_data[i].coins
# Gather the Probabilities of each Node
    Transacting
for i in range(len(transactor_data)):
 deck.append(Card(
    transactor_data[i].miner_address,
    transactor_data[i].coins
    /total_coins))
```

The second situation involves all the situations where the model has been initialized already. In this case we use a dithering algorithm[2n]. The algorithm basically subtracts a certain amount of probability from the previous occurring event, where the new P_{i} equates to:

$$P_{\text{i}} = Pr(E_{\text{i}}) - step$$

Where, once again,

$$P_{\text{i}} = \frac{C_{\text{i}}}{C_{\text{total}}}$$

and the $step$ is chosen by the programmer. We have chosen it be **0.1** through empirical calculations.

This results of this function is a 'deck' of events, which entail the probability of each transactor, transacting, abstracted to be an event E_{i}. In both situations the list of events is passed on to the function `deck_chooser`.

d) Deck Chooser: In order to induce true random selection, the probabilistic model involves the shuffling of the cards. However, the distribution of each shuffle has been visualized in the figures referenced below. If we divide the deck into three different stacks of cards. For each of the three decks, $f(x)$ can be equated to P_{o}. P_{o} is the probability of the a card from a particular stack being placed in position x within the stack.

For the first stack of the lowest $\frac{1}{3}$ of the cards, we can induce a standard deviation of **10** giving us the graph in **Figure 1**. Figure 2 and Figure 3 deal with the second and last $\frac{1}{3}$ of the cards.

Listing 24: Gaussian Based Random Selection

```
 list1 = deck[:len(deck)/3]
 list1 = random.sample(list1,len(list1))
 list2 = deck[len(deck)/3:2*(len(deck))]
 list2 = random.sample(list2,len(list2))
 list3 = deck[2*len(deck):]
 list3 = random.sample(list3,len(list3))
```

The Code above implements the model discussed in this section. Based on this shuffling, we maintain a general order of three sections of the population, while inducing an unbiased random selection from the topmost stack of

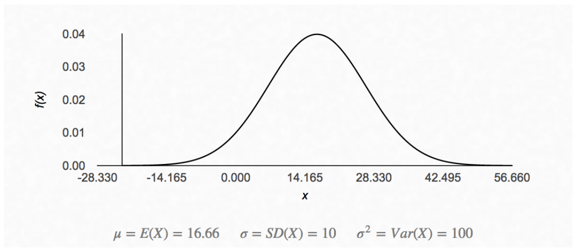

Fig. 1: The Probability Distribution for the first 33% of the cards

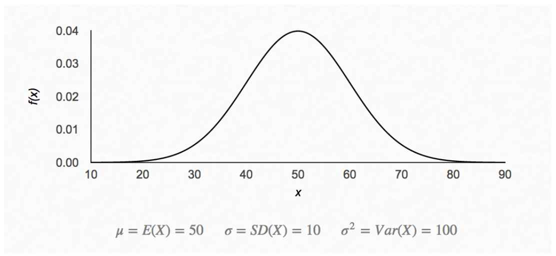

Fig. 2: The Probability Distribution for the second 33% of the cards

cards. Due to the dithering algorithm, there is a fair bit of migration between the stacks, creating a very robust model that mimics the nuances of a real socioeconomic population using a currency.

VII. Analysis

The task of analysis is quite difficult given the fact that we want to check the most optimal configuration for a miner given the fact that we have multiple constraints we want to randomize as given below:

- CPU configuration
- GPU configuration
- Hashing Function
- Number of Threads/Kernels within the CPU and GPU

The individual distribution of Energy Efficiency against the value of CPU configuration is given in 4

This would require an $O(n)$ crawl through all the different possible values for $\text{CPU}_{\text{config}}$. Were we to employ the algorithm below, we would get a good estimate of the graph as shown in 5.

One cannot see too much of a difference between the two from a holistic observation of the two graphs. In fact,

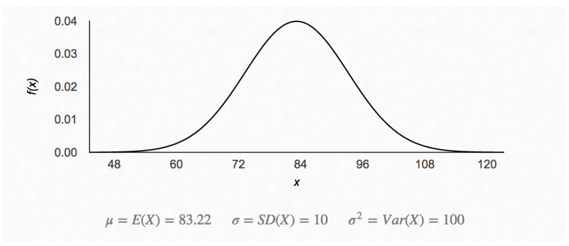

Fig. 3: The Probability Distribution for the last 33% of the cards

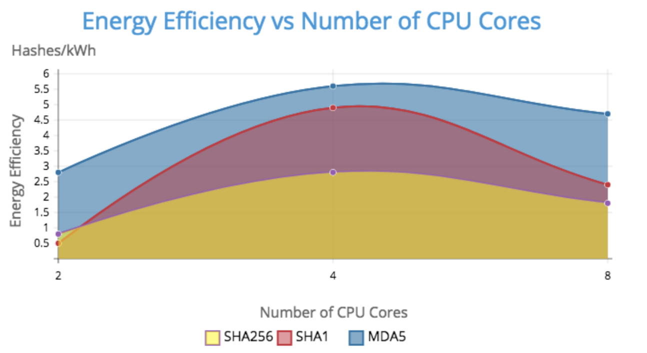

Fig. 4: The Values of Energy Efficiency given the CPU Configuration (No. of Cores) Original

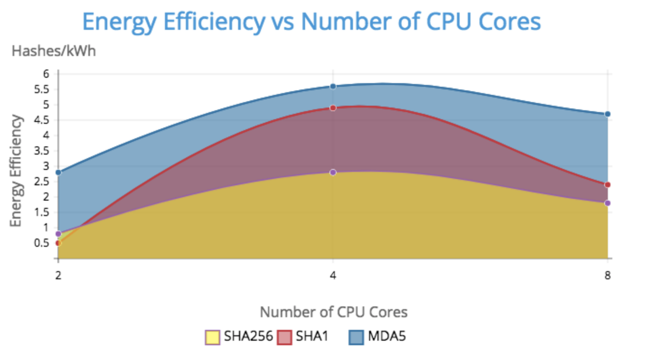

Fig. 5: The Values of Energy Efficiency given the CPU Configuration (No. of Cores) Sampled

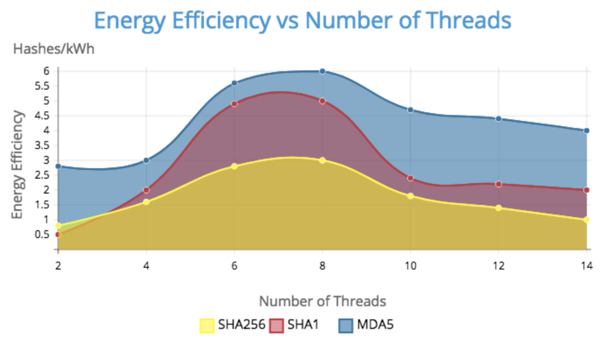

Fig. 6: The Values of Energy Efficiency given the CPU Configuration (No. of Cores) Original

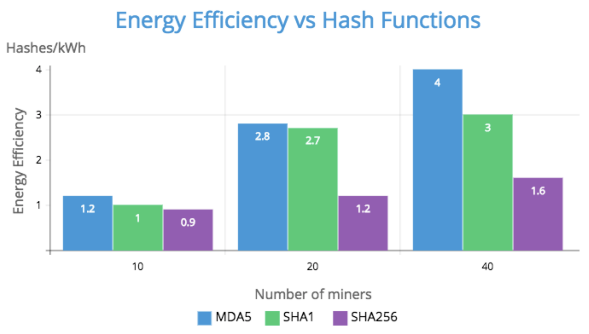

Fig. 7: The Values of Energy Efficiency given the CPU Configuration (No. of Cores) Original

the difference between the two, if we look at the sampled euclidean distance measure, it is 32.005. Before we proceed to the algorithm, the results for the Energy Efficiency that is varied according to the number of threads run is given in 6. Finally a summary of all the different results, with respect to each hashing algorithm is given in 7. One may notice that we have not given the results for the GPU variation of cores and its effect on the energy efficiency of the miners. This is due to the fact that using the GPU with *PyCUDA* slowed our system down and therefore did not render better results than that of the CPU. Thus, we did not include these results within our report. After further investigation, we discovered that the reason for this poor performance is due to the lack of support for bit operations, which are very crucial for the hashing process.

Our algorithm is as follows[19]:

- Randomly choose K values of constraint C greater than each other in a successive row where:

$$0 < C_0 < |max(C)|$$

$$C_{i-1} < C_i < |max(C)|$$

D_i is the data set being clustered.

- Figure out the Energy Efficiency for each of these C_i.
- Using this information, use **non-linear regression** to fit an exponential decay graph to the data, as shown in the `Python` code below:

```
params, cov = curve_fit(d0=guess,
   my_data[:,1], my_data[:,2],
      p0=guess)
```

Once we have the values of the individual energy efficiency distribution. We can take the value of of the parameter C_i where energy efficiency is the maximum and set it to to equal the highest probability (1 in this case). We can use this graph to scale the graph to that of a probability distribution.

After all the probability distributions of all the different parameters (C_i), all the probabilities are combined to form the **m dimensional joint probability distribution**. Since we cannot visualize an m dimensional graph (4D in our case, since we have four parameters that are being observed), the individual distributions are shown below. However, all of them are scaled to be probability distributions before they are combined to form the joint probability distribution.

We have the Random Variables $X_1, X_2, ..., X_m$, which correspond to each of the distributions. We can place linear restrictions on these Random Variables, determined by the constraints of our testing software and hardware. $K(x)$ is a membership indicator random variable, where $K = 1$ if $x \subset K$, and $K = 0$, if not. K is a convex body which defines the feasible region where the sampling may happen. A convex body is an n dimensional Euclidean space in a compact convex set with a non empty interior. A Euclidean space is a a geometric space where euclidean coordinates and distance can be applied.

K can also be thought as a hypercube $[0,1]^m$, without a loss of generality, and we create m-1 dimensional planes to sample our joint probability distribution.

Now we can recreate the joint probability graph through a constant number of samples, after which we can use non linear regression to fit an m dimensional curve to this data. Finding the global maximum for the energy efficiency would become a statistical problem, which was solved through the use of various Python libraries. Please refer to the paper presented in the references for a further understanding of this algorithm.

VIII. Conclusion

Now that we have completely and thoroughly conducted experimentation on the effect on the energy efficiency of blockchain miners against various parameters, we can make fair assumptions about their relation with miner energy efficiency.

We begin with the relation of CPU cores. We observe that its affect on energy efficiency increases with number of cores till a peak at 4 cores, after which energy efficiency values decline. The general graph for the three hashing functions show similar result with the md5 hashing being most energy efficient. This is due to the md5 being the least CPU intensive.

We found similar results for the relation between energy efficiency and CPU threads. However here the improvement in energy efficiency rapidly increased with increase in the number of threads. After a peak at 8 threads the efficiency values decline. The results for the three hashing functions show similar results as the cpu cores' analysis.

Through our experiment setup and research analysis, we have found that a miner running on a machine with 2 CPU cores, 4 threads and hashing with the MDA5 protocol was most energy efficient though it did not receive the most number of cryptocurrency tokens.

IX. Future Works

- In order to improve the realistic modeling of a cryptocurrency, it is essential that `F.Block` realize:

  $$Miners \subset Transactors$$

  As of right now the miners and the transactors are mutually exclusive.
- Testing our blockchain network with more devices connected. This allows us to analyze results for a wider range of hardware.
- We can use Ethereum which is a decentralized platform to build custom blockchain. It enables developers to create markets, store registries of debts or promises, move funds in accordance with instructions given long in the past (like a will or a futures contract) and many other things that have not been invented yet, all without a middle man or counterparty risk.

References

[1] Yli-Huumo, J., Ko, D., Choi, S., Park, S. and Smolander, K., 2016. Where Is Current Research on Blockchain Technology?A Systematic Review.PloS one,11(10), p.e0163477. [2] https://hbr.org/2017/01/the-truth-about-blockchain. (2017). [Blog].

[2] https://hbr.org/2017/01/the-truth-about-blockchain. (2017). [Blog].

[3] Large Purse Shop. (2017).What are Blockchains (Block Chains)?. [online] Available at: https://largepurseshop.com/blogs/the-residual-income-solution/what-are-blockchains-block-chains [Accessed 25 Oct. 2017].

[4] Blockgeeks. (2017).What is Blockchain Technology? A Step-by-Step Guide For Beginners. [online] Available at: https://blockgeeks.com/guides/what-is-blockchain-technology/ [Accessed 25 Oct. 2017].

[5] Wang, L. and Liu, Y., 2015, March. Exploring miner evolution in bitcoin network. InInternational Conference on Passive and Active Network Measurement(pp. 290-302). Springer, Cham.

[6] esotera. (2017).Thin Clients and Blockchain Explorers - esotera. [online] Available at: https://esotera.eu/clients-explorers/ [Accessed 25 Oct. 2017].

[7] Nakamoto, S., 2008. Bitcoin: A peer-to-peer electronic cash system.

[8] Floyd, D. (2017).51% Attack. [online] Investopedia. Available at: http://www.investopedia.com/terms/1/51-attack.asp [Accessed 25 Oct. 2017].

[9] Paul, G., Sarkar, P. and Mukherjee, S., 2014, December. Towards a more democratic mining in bitcoins. InInternational Conference on Information Systems Security(pp. 185-203). Springer, Cham

[10] Woodward, A. (2017).Preventing Selfish Mining In The Blockchain. [online] Profwoodward.org. Available at: https://www.profwoodward.org/2016/05/preventing-selfish-mining-in-blockchain.html [Accessed 25 Oct. 2017].

[11] En.bitcoin.it. (2017).Controlled supply - Bitcoin Wiki. [online] Available at: https://en.bitcoin.it/wiki/Controlled_supply [Accessed 25 Oct. 2017].

[12] Dev JA. Bitcoin mining acceleration and performance quantification. InElectrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on 2014 May 4 (pp. 1-6). IEEE.

[13] Lexology.com. (2017).The Fueling Of MLM through Bitcoins — Lexology. [online] Available at: https://www.lexology.com/library/detail.aspx?g=1babaa6e-b19c-4498-9012-5846beb31152 [Accessed 25 Oct. 2017].

[14] Barkatullah, J. and Hanke, T., 2015. Goldstrike 1: Cointerra's first-generation cryptocurrency mining processor for bitcoin.IEEE micro,35(2), pp.68-76.

[15] itarisWriter, B., Vitaris, B. and Vitaris, B. (2017).Visa Files Patent for Blockchain-Based Digital Asset Network. [online] Bitcoin Magazine. Available at: https://bitcoinmagazine.com/articles/visa-files-patent-blockchain-based-digital-asset-network/ [Accessed 25 Oct. 2017].

[16] Blockgeeks. (2017).What is Ethereum? A Step-by-Step Beginners Guide [Ultimate Guide]. [online] Available at: https://blockgeeks.com/guides/what-is-ethereum/ [Accessed 25 Oct. 2017].

[17] Gilbert, H. and Handschuh, H., 2003, August. Security analysis of SHA-256 and sisters. In International workshop on selected areas in cryptography (pp. 175-193). Springer, Berlin, Heidelberg. Vancouver

[18] Silva, Aristfanes Correia; Lucena, Paula Salgado; Figuerola, Wilfredo Blanco (13 December 2000). "Average Dithering". Image Based Artistic Dithering. Visgraf Lab. Retrieved 2007-09-10.

[19] Huang, Z. and Kannan, S., 2011, January. On Sampling from Multivariate Distributions. In APPROX-RANDOM (pp. 616-627).