CS550 Final Report: DataSys Coin Blockchain

Haoran Wang

A20449730

Abstract

In this project, we implemented a centralized blockchain called DataSyc Coin (DSC) using Java. This centralized blockchain is fully functioning with a command-line user interface. Overall, DSC contains 6 components: wallet client, blockchain server, pool server, validator client, and monitor server. We use sockets for client-server communication. Additionally, we implemented Proof of Storage (PoS) and solved unique implementation challenges that come with the Java language. Finally, we conduct a strong scaling experiment on the DSC system on 24 VMs and report the performance on latency and throughput for each type of the three validators.

1 Introduction

In 2023, a significant number of people are acquainted with the concept of blockchain, and many have likely come across the term Bitcoin. However, for those not well-versed in the subject, understanding how blockchain operates may be challenging. Essentially, in blockchain, data is stored in a distributed ledger, and the technology ensures integrity and availability, enabling participants to write, read, and verify transactions in the ledger. Notably, blockchain prohibits deletion and modification operations on recorded transactions and other ledger information. The system relies on cryptographic primitives and protocols, such as digital signatures and hash functions, to support and secure the blockchain, ensuring that recorded transactions are integrity-protected, authenticity-verified, and non-repudiated. Additionally, as a distributed network, blockchain requires a consensus protocol—a set of rules followed by every participant—to achieve a globally unified view and enable unanimous agreement on the ledger's content. [Zheng et al.(2018), Guo and Yu(2022)]

Interestingly, the original idea of blockchain was presented in the Bitcoin whitepaper [Nakamoto(2008)]. The paper, believed to be authored by an individual or group using the pseudonym Satoshi Nakamoto, introduced the concept of cryptocurrency and blockchain while contributing to the development of the initial Bitcoin software. As outlined in the white paper, the blockchain infrastructure was envisioned to facilitate secure peer-to-peer transactions, eliminating the need for reliance on trusted third parties like banks or governments. Despite widespread speculation, Nakamoto's true identity remains undisclosed, fueling various theories. [Wüst and Gervais(2018)]

In this project, we implemented a centralized blockchain called DataSyc Coin (DSC) using Java. The arrangement of this report is outlined as follows: Section 2 delves into the comprehensive examination of the six components constituting DSC. Section 3 details our approaches to implementing each component and explicates the method we employed for proof of storage. Section 4 elucidates our experimental settings and presents the results of the experiments. Ultimately, we encapsulate the pivotal discoveries of this project in the concluding summary in Section 5.

2 Problem Statement

DataSyc Coin (DSC) is a centralized blockchain system. As shown in Figure 1, it contains six unique components: wallet, blockchain, pool, metronome, validator, and monitor. The components communicate with each other using network sockets.

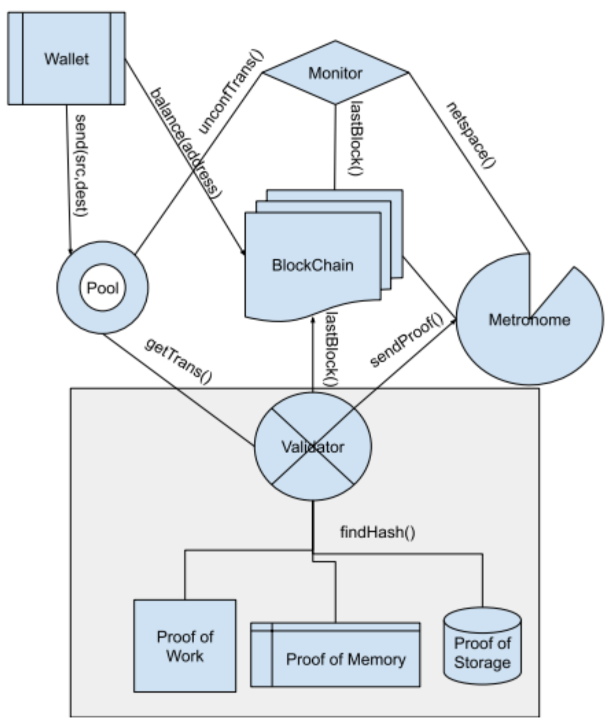


Figure 1: DataSys Coin Centralized Architecture.

The blockchain consists of a chain of blocks, where each block stores multiple transactions. The specification of a transaction and a block is shown in Figure 2. The difficulties just mean the number of bits that need to be matched exactly to the target hash.

- Transaction (128B)
  - Sender Public Address (32B)
  - Recipient Public Address (32B)
  - Value (unsigned double, 8B)
  - Timestamp (signed integer 8B)
  - Transaction ID (16B)
  - Signature (32B)
- Block (128B header + 128B*#trans) – multiple transactions will be stored in a block on the blockchain
  - Block Size (unsigned integer 4B)
  - Block Header (56B)
    - Version (unsigned short integer 2B)
    - Previous Block Hash (32B)
    - BlockID (unsigned integer 4B)
    - Timestamp (signed integer 8B)
    - Difficulty Target (unsigned short integer 2B)
    - Nonce (unsigned integer 8B)
  - Transaction Counter (unsigned integer 4B)
  - Reserved (64B)
  - Array of Transactions (variable)

Figure 2: Specifications of transaction and block.

Wallet is a client that creates wallets, send transactions, and view balance. It has a command-line interface and prints out the necessary information.

Blockchain is a server that stores blocks, and offers interfaces to interact with this blockchain, such as retrieving the last block header upon request, lookup the state of a transaction, lookup the balance of an address, etc.

Pool is a server that receives transaction, and create submitted and unconfirmed data structures that combine queue and hashmap.

Metronome is a server that has dynamic difficulty, creates an empty block every 6 seconds, accepts validators register, and reports statistics data.

Monitor is a server that simply collects statistics of the running system.

Validator is the main worker in the system. It verifies the transactions using three types of algorithm proof of work (PoW), proof of memory (PoM), and proof of storage (PoS).

3 Proposed Solution

In this section, we provide a detailed description of how we implement each of the six components and discuss the solutions for some of the unique challenges we faced when implementing in Java. Figure 3 shows the structure of our implementation. Generally, each component is contained in a Java file. `Helper.java` contains helper functions that are used repeatedly throughout the project.

```
project.
├── bitcoinj-core-0.16.jar(a library for working with Base58)
├── BlockChain.java(implement blockchain component)
├── Block.java(define a block)
├── client.sh(send transactions sequentially)
├── commons-codec-1.16.0.jar(a library for working with Blake3)
├── dsc-config.yaml(YAML style system configuration file)
├── dsc.java(command line interface)
├── dsc-key.yaml(YAML style save private key file)
├── dsc.sh(wrapper shell, used to simplify java command line)
├── evaluation
│   ├── merge.sh(merge and process experimental data)
│   ├── node.all(all 24 node IP)
│   ├── node.blc(blockChain Server IP)
│   ├── node.cli-1(wallet 1 client IP)
│   ├── node.cli-2(wallet 2 clients IP)
│   ├── node.cli-4(wallet 4 clients IP)
│   ├── node.cli-8(wallet 8 clients IP)
│   ├── node.mon(monitor Server IP)
│   ├── node.mtr(metronome Server IP)
│   ├── node.pol(pool Server IP)
│   ├── node.val(validator Server IP)
│   ├── start_vm.sh(Startup 24 vms)
│   └── stop_vm.sh(Stop 24 vms)
├── Helper.java(helper programme for SHA256 etc.)
├── Makefile(compile automation)
├── Metronome.java(implement metronome component)
├── Monitor.java(implement monitor component)
├── Pool.java(implement pool component)
├── README.md(this file)
├── snakeyaml-2.2.jar(a library for working with YAML)
├── Transaction.java(define a transaction)
├── Validator.java(implement validator component)
└── Wallet.java(implement wallet component)
```

Figure 3: Project structure.

3.1 Wallet

The main challenge of implementing wallet-create is to use SHA256 to create public/private keys of 256-bit length. To do so, we first use `java.security.KeyPairGenerator` to generate 256-bit public/private key-pairs using *Elliptic Curve (EC)* algorithm as the signature.

```
ECGenParameterSpec ecSpec = new ECGenParameterSpec( stdName: "secp256k1");
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance( algorithm: "EC");
keyPairGenerator.initialize(ecSpec, new SecureRandom());
KeyPair keyPair = keyPairGenerator.generateKeyPair();

pubKey = keyPair.getPublic();
privKey = keyPair.getPrivate();

String pub_Hex = Helper.bytesToHex(pubKey.getEncoded());
String priv_Hex = Helper.bytesToHex(privKey.getEncoded());
```

Next, we convert the generated public-private key pairs, which are byte arrays into Hex strings, using `org.apache.commons.codec.binary.Hex` library.

```
String pub_Hex = Helper.bytesToHex(pubKey.getEncoded());
String priv_Hex = Helper.bytesToHex(privKey.getEncoded());
```

```
public static String bytesToHex(byte[] b) {
    return String.valueOf(Hex.encodeHex(b, toLowerCase: true));
}
```

Then, we use SHA256 to hash the obtained Hex strings, and convert it to *Base58* encoding.

```
pubHashed = Helper.sha256(pub_Hex);
privHashed = Helper.sha256(priv_Hex);

UUID uuid = UUID.randomUUID();
fingerprint = uuid.toString();
```

```
public static String sha256(String string) throws NoSuchAlgorithmException {
    string = "80" + string;
    byte[] data = hexToBytes(string);
    byte[] digest = MessageDigest.getInstance( algorithm: "SHA-256").digest(data);

    return Base58.encode(digest);
}
```

Finally, we set the permission of `dsc-key.yaml` to 400. Also, we check if the file already exists, and abort if so. The code snippets are shown below.

```
Path path = Paths.get( first: "./dsc-key.yaml");
Set<PosixFilePermission> perms = PosixFilePermissions.fromString( perms: "r--------");
Files.setPosixFilePermissions(path, perms);
```

```
if (f.exists() && !f.isDirectory()) {
    System.out.println(Helper.get_timestamp() + " DSC v1.0");
    System.out.println(Helper.get_timestamp() + " Wallet already exists at dsc-key.yaml, wallet create aborted");
    System.exit( status: 1);
} else {
```

For wallet-send, we simply assign a random 16B transaction ID and then make a request to the pool server. The pool server then responds with an acknowledgment if it receives the request.

```
void send(double coin, String dest) throws IOException, NoS
    Random rd = new Random();
    byte[] txID = new byte[16];
    rd.nextBytes(txID);

    String txIDStr = Base58.encode(txID);
    String signStr = txIDStr + get_pubKey() + dest + coin;
```

3.2 Blockchain

To create the genesis block, we create a new `block` object and set the previous hash to a new 32B array.

```
public Block create_genesis_Block() {
    LinkedList<Transaction> txs = new LinkedList<>();
    byte[] prev_hash = new byte[32];
    Long timestamp = Instant.now().getEpochSecond();
    Block genesis_block = new Block(txs, prev_hash, block_id: 0, timestamp, (short) 30, nonce: 0);
    return genesis_block;
}
```

We represent the blockchain as a linked list and use two hash maps named `confirmed` and `empty` to represent the confirmed block and empty block respectively. The *key* represents the block ID, and the *value* represents the index in the linked list.

```
public static LinkedList<Block> blockChain = new LinkedList<~>();
1 usage
public static HashMap<Integer, Integer> confirmed = new HashMap<>();
3 usages
public static HashMap<Integer, Integer> empty = new HashMap<>();
```

3.3 Pool

The implementation of the pool server is straightforward. The implementation details are trivial.

3.4 Metronome

We implemented the metronome server with dynamic difficulty as required. When the number of validator workers is less than 4, we decrease the difficulty by 1. When the number of validator workers is larger than 8, we increase the difficulty by 1.

```
public int calculate_diff() {
    int vailidator_num = validators.size();
    if (vailidator_num < 4)
        return difficulty - 1;
    else if (vailidator_num > 8)
        return difficulty + 1;
    else
        return difficulty;
}
```

3.5 Validator

We implemented all three types of validators: PoW, PoM, and PoS.

3.5.1 PoW

For PoW implementation, we strictly follows the given pseudo code.

```
public long pow_lookup(String prefix_hash_lookup) {
    double start_time = System.currentTimeMillis();

    counter = 0;
    while (System.currentTimeMillis() < (start_time + Helper.block_time)) {
        String hash_input = this.fingerprint + this.public_key + Long.toString(this.NONCE);
        byte[] hash_output_byte = Helper.blake3(hash_input);
        synchronized (lock) {
            counter++;
        }
        String prefix_hash_output = Helper.ByteArraysToBinary(hash_output_byte).substring(0, this.difficulty);

        if (prefix_hash_lookup.equals(prefix_hash_output))
            return this.NONCE;
        else
            synchronized (lock) {
                this.NONCE++;
            }
    }
    return -1;
}
```

We use Blake3 hash from `commons-codec-1.16.0.jar` library.

```java
public static byte[] blake3(String hash_input) {

    Blake3 hasher = Blake3.initHash();
    hasher.update(hash_input.getBytes(StandardCharsets.UTF_8));
    byte[] hash = new byte[24];
    hasher.doFinalize(hash);
    // return bytesToHex(hash);
    return hash;
}
```

3.5.2 PoM

To avoid multiple threads accessing the same section in `memory_store`, first split `memory_store` into different sections corresponding to the number of threads. Then, one thread only works on one specific section.

```java
int START = Integer.valueOf(Thread.currentThread().getName()) * this.num_hashes;
int END = START + this.num_hashes;
```

```java
int size = (1024 * 1024 * 1024) / (24 + 8);
int num_hashs = size / threads_hash;
memory_store.createMemoryStore(size);
```

We use `Arrays.sort()` from Java's Arrays library to sort. Then we implemented binary search to search.

```java
public static long lookupMemoryStore(String prefix_hash_lookup, int difficulty, int size) {

    int left = 0;
    int right = size;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        byte[] hash_output = Arrays.copyOfRange(memory_store[mid], from: 0, to: 24);
        byte[] nonce = Arrays.copyOfRange(memory_store[mid], from: 24, to: 32);
        String prefix_hash_output = Helper.ByteArraysToBinary(hash_output).substring(0, difficulty);

        // Check if prefix_hash_lookup is present at mid
        if (prefix_hash_output.compareTo(prefix_hash_lookup) == 0)
            return Helper.bytesToLong(nonce);

        // If prefix_hash_lookup greater, ignore left half
        if (prefix_hash_output.compareTo(prefix_hash_lookup) < 0)
            left = mid + 1;

            // If prefix_hash_lookup is smaller, ignore right half
        else
            right = mid - 1;
    }

    // If reach here, then element was not present
    return -1;
}
```

3.5.3 PoS

Figure 4 shows an illustration of our PoS implementation. The cups are represented as `byte[][][]` array in Java, and the buckets are stored as a file for each bucket. We store 256 buckets, with 40 cups per bucket. The cup size is 32,768. Each cup has 32B. Therefore, the total storage is $256 * 40 * 32768 * 32 = 10737418240$ bytes, which is 10GB.

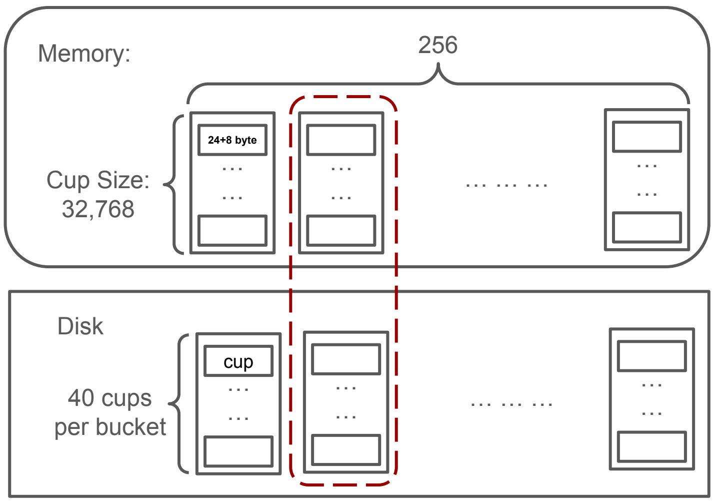


Figure 4: Illustration of PoS implementation.

We encountered a unique challenge when implementing the function to locate the bucket. Unlike C, Java does not have unsigned integers. The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive). To solve this issue, we simply do a bit-wise & operation with 0xFF to convert it into a range from 0 to 255.

```
byte prefix_hash_output = hash_output_byte[0];
byte[] nonce_byte = Helper.longToBytes(this.NONCE);

int bucket_num = prefix_hash_output & 0xFF;
```

We use `RandomAccessFile` function to write to buckets, which requires `seek(position)` to find the position to write to.

```java
public void pos_write(byte[][] buffer, int bucket_num, int cup_no) throws IOException {
    long position = cup_no * (this.cup_size * 32);

    File bucket_name = new File( pathname: this.vaultFile + "/" + "bucket" + String.format("%03d", bucket_num));
    RandomAccessFile raf = new RandomAccessFile(bucket_name,  mode: "rw");
    raf.seek(position);
    for (int i = 0; i < this.cup_size; i++) {
        raf.write(buffer[i]);
    }
    raf.close();
}
```

However, the largest position, which is $256 * 40 * 32768 * 32$ is too large, and in Java, this number becomes negative. The experiment below shows that this number is negative in Java. To solve this issue, we divide the buckets into 256 files.

```java
public static void main(String[] args) throws IOException {
    long l_1 = 256 * 40 * 32768 * 32;
    long l_2 = 40 * 32768 * 32;
    System.out.println("l1 = " + l_1);
    System.out.println("l2 = " + l_2);
}
```

```
(base) → bruce@thebeast  ~/work/project  java tt.java
l1 = -2147483648
l2 = 41943040
```

We store 256 buckets under `dsc-pos.vault` folder.

```
(base) → bruce@thebeast  ~/work/project  cd dsc-pos.vault
(base) → bruce@thebeast  ~/work/project/dsc-pos.vault  ls
bucket000  bucket026  bucket052  bucket078  bucket104  bucket130  bucket156  bucket182  bucket208  bucket234
bucket001  bucket027  bucket053  bucket079  bucket105  bucket131  bucket157  bucket183  bucket209  bucket235
bucket002  bucket028  bucket054  bucket080  bucket106  bucket132  bucket158  bucket184  bucket210  bucket236
bucket003  bucket029  bucket055  bucket081  bucket107  bucket133  bucket159  bucket185  bucket211  bucket237
bucket004  bucket030  bucket056  bucket082  bucket108  bucket134  bucket160  bucket186  bucket212  bucket238
bucket005  bucket031  bucket057  bucket083  bucket109  bucket135  bucket161  bucket187  bucket213  bucket239
bucket006  bucket032  bucket058  bucket084  bucket110  bucket136  bucket162  bucket188  bucket214  bucket240
bucket007  bucket033  bucket059  bucket085  bucket111  bucket137  bucket163  bucket189  bucket215  bucket241
bucket008  bucket034  bucket060  bucket086  bucket112  bucket138  bucket164  bucket190  bucket216  bucket242
bucket009  bucket035  bucket061  bucket087  bucket113  bucket139  bucket165  bucket191  bucket217  bucket243
bucket010  bucket036  bucket062  bucket088  bucket114  bucket140  bucket166  bucket192  bucket218  bucket244
bucket011  bucket037  bucket063  bucket089  bucket115  bucket141  bucket167  bucket193  bucket219  bucket245
bucket012  bucket038  bucket064  bucket090  bucket116  bucket142  bucket168  bucket194  bucket220  bucket246
bucket013  bucket039  bucket065  bucket091  bucket117  bucket143  bucket169  bucket195  bucket221  bucket247
bucket014  bucket040  bucket066  bucket092  bucket118  bucket144  bucket170  bucket196  bucket222  bucket248
bucket015  bucket041  bucket067  bucket093  bucket119  bucket145  bucket171  bucket197  bucket223  bucket249
bucket016  bucket042  bucket068  bucket094  bucket120  bucket146  bucket172  bucket198  bucket224  bucket250
bucket017  bucket043  bucket069  bucket095  bucket121  bucket147  bucket173  bucket199  bucket225  bucket251
bucket018  bucket044  bucket070  bucket096  bucket122  bucket148  bucket174  bucket200  bucket226  bucket252
bucket019  bucket045  bucket071  bucket097  bucket123  bucket149  bucket175  bucket201  bucket227  bucket253
bucket020  bucket046  bucket072  bucket098  bucket124  bucket150  bucket176  bucket202  bucket228  bucket254
bucket021  bucket047  bucket073  bucket099  bucket125  bucket151  bucket177  bucket203  bucket229  bucket255
bucket022  bucket048  bucket074  bucket100  bucket126  bucket152  bucket178  bucket204  bucket230
bucket023  bucket049  bucket075  bucket101  bucket127  bucket153  bucket179  bucket205  bucket231
bucket024  bucket050  bucket076  bucket102  bucket128  bucket154  bucket180  bucket206  bucket232
bucket025  bucket051  bucket077  bucket103  bucket129  bucket155  bucket181  bucket207  bucket233
(base) → bruce@thebeast  ~/work/project/dsc-pos.vault
```

Each file is 40MB. Some files are 41MB due to unbalanced hash.

```
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket223
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket224
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket225
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket226
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket227
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket228
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket229
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket230
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket231
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket232
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket233
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket234
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket235
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket236
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket237
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket238
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket239
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket240
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket241
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket242
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket243
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket244
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket245
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket246
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket247
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket248
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket249
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket250
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket251
-rw-rw-r-- 1 bruce bruce 40M Nov 29 15:42 bucket252
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket253
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket254
-rw-rw-r-- 1 bruce bruce 41M Nov 29 15:42 bucket255
(base) → bruce@thebeast  ~/work/project/dsc-pos.vault
```

We make system call in Java to sort the files.

```java
public void pos_sort() throws IOException, InterruptedException {
    ProcessBuilder builder = new ProcessBuilder();
    builder.directory(new File(new String(this.vault)));
    for (int i = 0; i < 256; i++) {
        File bucket_name = new File(vault + "/" + "bucket" + String.format("%03d", i));

        builder.command("sh", "-c", "sort " + bucket_name);
        Process process = builder.start();

        boolean isFinished = process.waitFor(600, TimeUnit.SECONDS);
        if (!isFinished) {
            process.destroyForcibly();
        }
    }
}
```

3.6 Monitor

The implementation of the monitor server is straightforward. The implementation details are trivial.

4 Evaluation

We conducted a strong scaling experiment using 24VMs. We conducted strong scaling experiments on both latency and throughput.

4.1 Latency

We send 128 transactions sequentially and wait for each one to be confirmed on the blockchain. The latency is the time difference from submit to confirm. We compute the average, minimum, and maximum latency, and report for each scale, 1, 2, 4, and 8 clients.

4.2 Throughput

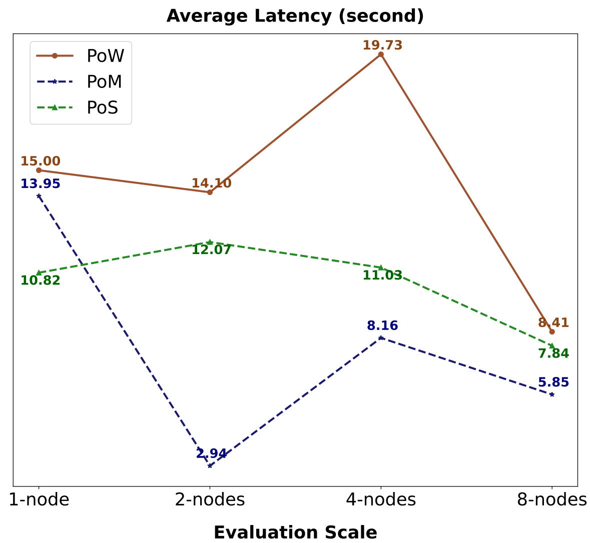
We conducted a strong scaling experiment on throughput where the benchmark client will send 128000 transactions sequentially, and after they are all submitted, wait for each one to be confirmed on the blockchain. The total time is the time from submit of the first transaction to confirmed. We compute the throughput by taking the total number of transactions and dividing it by the time of the experiment. We report the throughput for each scale, 1, 2, 4, and 8 benchmark clients.

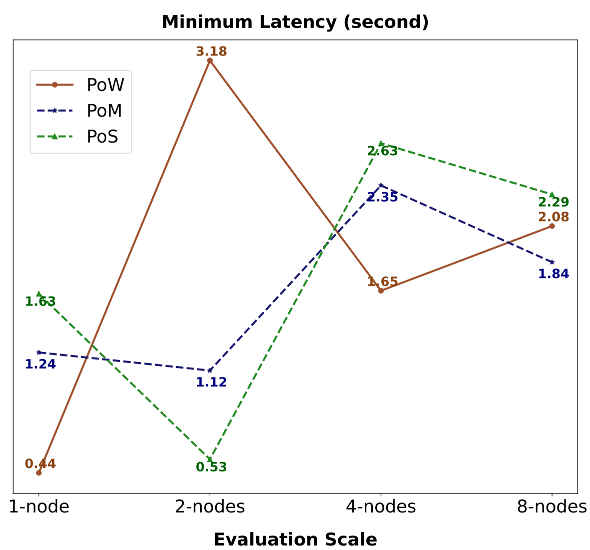
4.3 Experiment Results

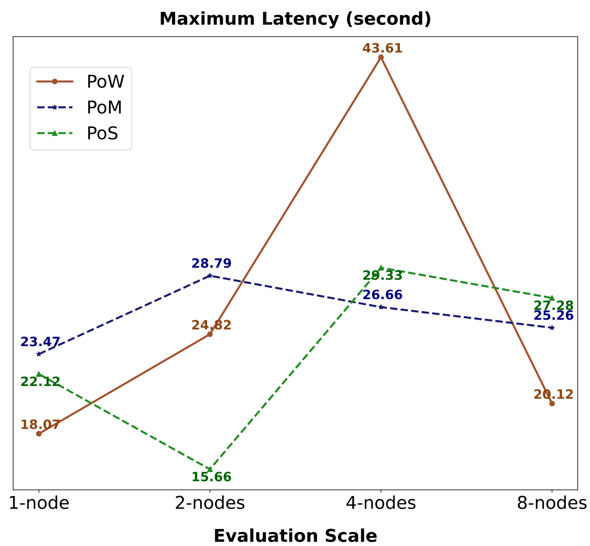
We report the experiment results in the table below:

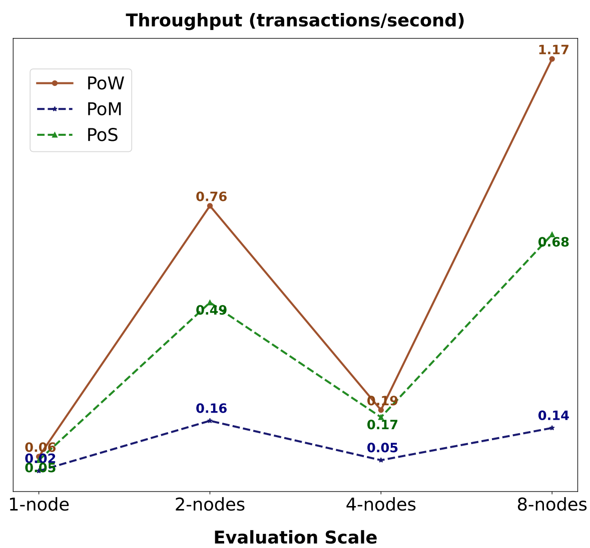
		Evaluation											
		1 client			2 clients			4 clients			8 clients		
		PoW	PoM	PoS	PoW	PoM	PoS	PoW	PoM	PoS	PoW	PoM	PoS
Latency	average	15.00	13.95	10.82	14.10	2.94	12.07	19.73	8.16	11.03	8.41	5.85	7.84
	minimum	0.44	1.24	1.63	3.18	1.12	0.53	1.65	2.35	2.63	2.08	1.84	2.29
	maximum	18.07	23.47	22.12	24.82	28.79	15.66	43.61	26.66	29.33	20.12	25.26	27.28
Throughput	throughput	0.06	0.02	0.05	0.76	0.16	0.49	0.19	0.05	0.17	1.17	0.14	0.68

We plot the experiment results in the figures below:









5 Conclusions

In this project, we implemented a centralized blockchain called DataSyc Coion. Specifically, we implemented all six components of the system and all three types of validators: proof of work, proof of memory, and proof of storage. We conducted strong scaling experiments on 24 VMs and reported the experiment results.

References

[Guo and Yu(2022)] Huaqun Guo and Xingjie Yu. 2022. A survey on blockchain technology and its security. *Blockchain: research and applications* 3, 2 (2022), 100067.

[Nakamoto(2008)] Satoshi Nakamoto. 2008. Bitcoin whitepaper. *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)* (2008).

[Wüst and Gervais(2018)] Karl Wüst and Arthur Gervais. 2018. Do you need a blockchain?. In *2018 crypto valley conference on blockchain technology (CVCBT)*. IEEE, 45–54.

[Zheng et al.(2018)] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. 2018. Blockchain challenges and opportunities: A survey. *International journal of web and grid services* 14, 4 (2018), 352–375.