# DOSP Fall 2022

## Project 1 - Bitcoin Miner

## Implementation

### Group Members:


1. Ayesha Naikodi, ma.naikodi@ufl.edu
2. Siju Sakaria, siju.sakaria@ufl.edu


## Usage

1. cd to the `src` folder
2. Start an Erlang shell node on one machine(server)
3. Start an Erlang shell node on another machine (worker)
4. Compile the erl file, `minebc.erl`
5. Start the server and then start the worker passing in the IP address of the server
6. To abort the mining process, call `stop` function

## Implementation Details

On the server, start the node
`erl -name nodename@ip_addr1 -setcookie .erlang.cookie` On the client, run
`erl -name nodename@ip_addr2 -setcookie .erlang.cookie`

Start the server using

```
minebc:startServer(k).
```

On the worker, connect to the server and join the mining using

```
minebc:startWorker('nodename@ip_addr').
```

To stop mining, run `minebc:stop().`

where k=number of leading zeros for the generated bitcoins required
ip_addr= IP Address of the server you want to connect to, in this case the local machine acts aa a miner

## Work Unit Generation

The string generation in our project for bitcoin mining is an iterative approach where we are generating a workload of 10,000,000 for each process on the machine. Each process will be provided a start number and workload. The process will then mine for the bitcoin between start number and the workload.

## Server work unit metrics

Here we have defined the number of processes to run as = No.of Cores * 4. This ensures that all the cores are used efficiently to mine bitcoins in a faster manner.

## Miner work unit metrics

Here we have defined the number of processes to run as one when the miner connect to the server. We can extend it to have multiple process on the same miner.

## Client Server Architecture

When we run the code as server we run No.of Cores * 4 on the server to ensure that all the cores are utilised completely to mine bitcoins based on the k values passed. Since we have a fixed workload, when a process in the server completes a given workload, it send a requests to server for the new workload. The server then allocates the new workload for this process. This mechanism repeats itself until the user manually kills the server.

When the client joins the server,it requests the server for the workload and the k value. Based on the workload and the k value, it allocates a single process for the bitcoin mining in the client.

When the server or client get a bitcoin, they send the value of the random string and its hash value back to the server to be printed on the console.

Note: When the server shuts down the client will not shutdown but it will throw a bad arg

exception.

# Assignment Details

## Work Unit

The work unit we defined for each actor was 10,000,000. We specifically chose this work unit because

a) This will avoid the possibility of repeated generation of the same string across the workers. b) Since different workers get different workloads-> this approach can be horizontally scalabale c) Better range for handling the mining operation for a given process/

## Result for startServer(5)

### Input

```
minebc:startServer(5).
```

### Output

```
<0.93.0>
"ma.naikodikjCdwMK0r"    "00000856e26c831d9304a2c548495ca04c5abe0ec7903a7e6
"ma.naikodiUhsGm86xP"    "00000daa5fef4ab04ea4971428665448cdee898740484c667
"ma.naikodiiUgiWUiv7Cx"          "00000d762119c136cf8409a4e1e4db36178d5b3f9
"ma.naikodibc8hn8"       "00000e7e6d5f158eaf9f18e8bb4aeee6a4571388667f57e5f
"ma.naikodixBF5xtb4"     "000005b68bc7254f9f0e7c856b653373c006f12d61d799145
"ma.naikodiQBquZU9hZvW"          "00000dd5a6d5eb11289d65adbca817c16584baa95
"ma.naikodimXn3Fxkd"     "0000027be80acf9d7b58c5172a0c23af6e4467998d4ddb0fb
"ma.naikodiTlZvWL"       "000004d65908a89ff0e1d89da132b715f4353bb794f01a537
```

Total CPU Time = 63.203 s
Total Real Time = 16.324 s
The ratio of CPU to Real Time = 3.872

## The coin with the most number of leading 0s that we were able to mine

7

### Input

```
minebc:startServer(7).
```

### Output

```
<0.88.0>
"ma.naikodivfzET"        "0000000652162cefe535720aa7769c4c382f75a441d26a9f4
```

### Largest number of working machines we tested our code on

We connected 2 machines, where Windows i5 octa core machine was the server with all cores utilized at 100% and 1 miner running on Mac M2 with one process running on them and utilising only one cpu 100% as we are running only one process on the client.