**Distributed Operating Systems Project 1**

**Team members:**

Sankalp Pandey – 92878142 (sankalppandey@ufl.edu)

Tushar Ranjan – 45562694 (tranjan@ufl.edu)

**Content:**

* DOSPProject1.zip
  + server.fsx
  + client.fsx
* ReadMe

**Brief description:**

The aim of the project is to mine bitcoins using the distributed systems. Bitcoin mining using SHA256 hashing is a heavy computational process that consumes plenty of time but by implementing actor-model framework we can split the workload and use multiple cores for faster computation.

**Steps to Run:**

* To run the server type: *dotnet fsi server.fsx {k}*
  + k – is a command line argument that takes the no. of leading zeroes
* To run the client type: *dotnet fsi client.fsx {k} {server’s IP} {port}*
  + k – is a command line argument that takes the no. of leading zeroes
  + Server’s IP – address of host machine
  + Port – port at which server is running

**Design**:

Server-

1. On running server ‘ServerActor’ is created and ‘start’ message is sent to ‘boss’ actor.
2. Boss actor creates multiple worker actors and divides the workload among them.
3. Server actors will start the computation as well as also handle the responses coming from the client actors parallelly.
4. Once the server finishes its computation and receives the message ‘client done’ from the client actor, it terminates the program.

Client-

1. On running client ‘Boss’ actor is created, and boss ‘ServerActor’ is selected using the IP address and port entered.
2. Client’s boss actor creates multiple worker actors and divides the workload among them.
3. Each client actor will then perform the computation and send the generated bitcoin to server.
4. Once the client’s computation is done it will send an ‘Client Done’ message to the server.

**Systems used:**

1. Apple M1 – 8 core machines
2. Apple M1 – 8 core machines

**Observation**:

1. We have created eight workers each, in server as well as client machine. The total workload is divided equally among all the workers, each worker takes a random string appended to gator username and computes the SHA256 hash until it finds a hash with the required no. of leading zeroes. Let’s say, the total workload = N, so each worker receives N/8 unit of work, which they work upon asynchronously.
2. Below is a comparison of Real time and CPU time with different values of k and no. of actors.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K (no. of zeroes) | No of actors | Real time | CPU time | Ratio |
| 2 | 8 | 00:03.644 | 00:01.062 | 0.291 |
| 4 | 8 | 00:06.183 | 00:12.562 | 2.031 |
| 16 | 00:01.833 | 00:08.515 | 4.645 |
| 32 | 00:05.113 | 00:25.656 | 5.021 |
| 5 | 8 | 01:38.872 | 07:40.531 | 5.362 |

**Result** - For k = 4, with 8 cores the ratio comes out to be **2.031**. This shows that the computation is being done parallelly, also with increasing the actors we notice that the ratio’s difference from 1 also increases.

**Snippet for 8-actors, k = 4 with ratio = 2.031:**

Activity monitor-

A picture containing table

Description automatically generated

Server console (k=4):

Text

Description automatically generated

Client console (k=4, IP= 10.20.33.15 port=9002):

Text

Description automatically generated

Bitcoins that we were able to mine for max value of k = 6

Text

Description automatically generated