Distributed Operating System Project 1

Kasiviswanathan Srikant Iyer-(52222519) Swaathi Reena Velavan - (12308520)

September 25, 2021

1 Problem Statement

We are given a computationally intensive job of finding bitcoins. We are given an input that specifies the desired number of leading zeros. We should generate random strings, compute SHA256 hash of the string and check the hexadecimal version of the hash, if it has the required number of leading zeros. We should also extend the project to allow distributive implementation. We should make use of F and the actor concurrency model.

2 Required Input

Required Input -

To run the program we require the following variables for implementation of the code -

1. N - Number of random values needed to be checked
2. k - Number of leading 0s to be checked
3. IPaddress - IP address of the server we need to connect to
4. Port - Port that is used for communication

3 Steps to Run

1. To setup the server, find the systems IP address and replace it in the file named "Server.fsx".
2. Start the server by typeing in the terminal "dotnet fsi Server.fsx 1000000 4", where values 1000000 denotes the number of hashes to be checked and 4 is the number of leading 0s to z/aqbe found.
3. Once the server is operational, note down the IP address and port number from the terminal in server window.
4. To start the client, type " dotnet fsi Client.fsx ¡servers IP address¿ ¡port number¿" and hit enter.
5. Note: If the client is not connected to the server within the assigned time (in our case 10 seconds) the server will start executing by itself till a connection is received.

4 Output

4.1 Observations

1. The number of actors to be created depends on the number of cores available in the systems. If the client has 8 cores and the server has 4 cores, actors are created in multiples of 12 and divided among the systems as per the number of cores.
2. Depending on the number of hashes to be checked the ratio of Runtime vs CPU time changes.
3. The client server requires inputs while running to know the IP address and port it should use to communicate, but once executed, it does not print any value.

4. The server requires 2 inputs which will tell the system how many values need to be checked and for how many leading 0s to look for. Once compiled the server will print out its found hashes and the hashes found by the client if there is a client connected.

5 CPU Utilization and Performance

Input 1:
N = 100000000 (10^8)
k = 4
Bitcoins Found = 1123
Real: 00:05:16.643, CPU: 00:25:41.438, GC gen0: 45247, gen1: 52, gen2: 4
CPU/Real: 4.9
Observation: As the workload increases, we notice an increase in CPU utilization across cores.

Input 2:
N = 1000000 (10^6)
k = 4
Bitcoins mined = 22
Real: 00:01:54.341, CPU: 00:05:09.468, GC gen0: 55743, gen1: 12, gen2: 1
CPU/Real: 3.34
Observation: When the workload is relatively small, the CPU utilization across cores is quite less averaging around 18% as shown in Figure 3.

Input 3:
N = 10000 (10^4)
k = 4
Bitcoins mined = 0
Real: 00:00:10.560, CPU: 00:00:15.234, GC gen0: 1845, gen1: 2, gen2: 0
CPU/Real: 1.44
Observation: The aprobability of mining bitcoins is less when the workload is less. It increases as we increase the workload and can be noticed from Input 1 and Input 2. Although, the tasks were divided among actors, we did not find a bitcoin (see Figure 2).

6 Findings

- The coin with the most 0s you managed to find: **7**.
- The largest number of working machines you were able to run your code with: **2**.

References

[Gre93] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.

https://docs.microsoft.com/en-us/dotnet/fsharp/tour

https://fsharp.org/guides/web/

https://chester.codes/games-room-fsharp-signalr-akka-net

https://www.rickyterrell.com/

https://fsharp.org/

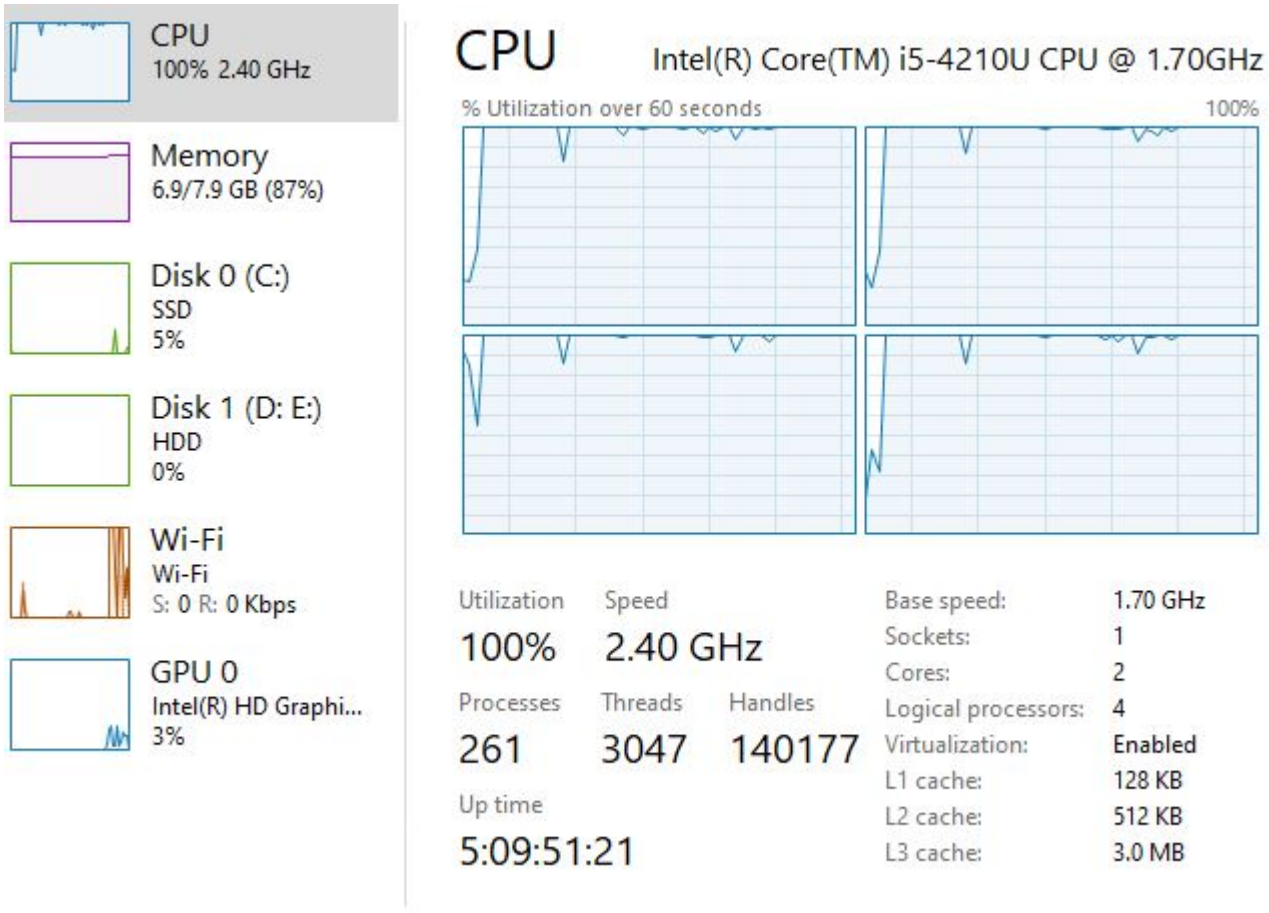

Figure 1: CPU cores being used.

```
Found in Server "0000B04EAA1AEB49477A5A3A69954CCED446F6794EADF4C8EA274CCFBBE75B12"
Found in Server "0000025A48DBFEEC78C9A267DD7C772D40A02F3F395758D592AA336FD3767674"
From Client : Client- 0000885BC82A469BF77E7307683E1CB1E62009F3EF80260F1A422C718E311ED4
From Client : Client- 00004F0453CACB172F2C3ED8ACF08F174CA56C2975F071CCF76C2716B7E1D528
From Client : Client- 000016446F60E337960FCA191E7DC593A45192F2CB9A9ACFA359B5670293FD94
From Client : Client- 00004FE6B75D8A1FD024699B283C0A2C52EF08987249E1E9C411F86A33317AA5
From Client : Client- 000014D2720D2AD3F2F9063608B07D8A0EDADB25F79F6A3102DB71DAA8E91244
From Client : Client- 00008C561E12C25697F05743D38553D2F377B9270CE24C082C24A9485280D622
[INFO][24-09-2021 04:45:19][Thread 0008][akka://RemoteFSharp/user/localDisp] Message [S
spatcher#1317102886] to [akka://RemoteFSharp/user/localDisp#1566646979] was unhandled.
f or adjusted with configuration settings 'akka.log-dead-letters' and 'akka.log-dead-le
[INFO][24-09-2021 04:45:20][Thread 0016][akka.tcp://RemoteFSharp@192.168.0.206:8777/sys
FClientFsharp%40localhost%3A8777-1] Removing receive buffers for [akka.tcp://RemoteFSha
8777]
Real: 00:00:10.560, CPU: 00:00:15.234, GC gen0: 1845, gen1: 1, gen2: 0
PS C:\F#\Net>
```

Figure 2: CPU Utilization for input 3.

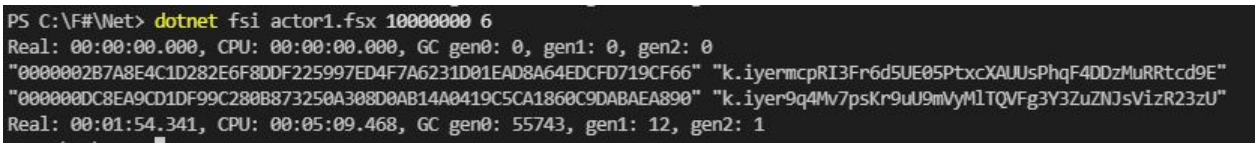

Figure 3: CPU Utilization for input 2.