DOS Project 1

Rupayan Das - 63992113
Pirouz Naghavi - 30158432

Getting Started

Requirements

- **.NET 5 SDK on both machines**

Running the project on a single machine

- *dotnet fsi ProjectPart1.fsx <numberOfLeadingZeroes>*
- eg: dotnet fsi ProjectPart1.fsx 4
- UF CISE COP5615 Project 1 Part 1 - YouTube

Running the project on a local and remote machine

First the server needs to start running
- Before running, in the config, the ipAddress has to be changed to the machine's IP
- *dotnet fsi server002.fsx*

After the server has started running, we can run the client
- *dotnet fsi client002.fsx <numberOfLeadingZeroes> <IPAddressOfServer> <PortNumber>*
- eg: dotnet fsi client002.fsx 4 172.16.107.143 8777

Working Demo - UF CISE COP5615 Project 1 Part 2 - YouTube

Some of the SHA256 hash values our program produced:

Leading Zeros	String Passed to SHA256	SHA256 Value
4	rupayandas;0axphefc4z	0000c3756a8f7f65fd9ebd4203f533a8411280f19ea73400609bf391c7826867
5	rupayandas;sxltqv8mh0	000006df3cbe6a29681dca0e146d8ead6fe6fb90fe4538f212b8dcfc011cf17c
6	rupayandas;04rsj8mw3x	0000003fa9381da54d17c26a2c93a1f1e2e1ee89543358397a26b74e438098de
7	rupayandas;7ipojuirq7	0000000d9b0616043f15db3ec7efb526522a3d8d793a065a109ea2d95e2c1320
8	rupayandas;qbqm353kpf	0000000021175debdc235b8e1a7ca70b29edd1f70016348b80622d9dec4ae807

Project Questions

Size of The Work unit

The algorithm starts by **generating 6 workers on both client and server** initially but after each worker's task is done the dispatcher will generate a new worker to take its place. The size of work done by each worker was determined by an algorithm that runs stablly on our machines and doesn't run into AKKA actor model timeout issues.

For this section we realized that sometimes our random string generation code can be very unlucky and even after a very long period of time of guessing random strings and hashing them, the algorithm was unable to find any useful hash values with large numbers, for example four, of leading zeros.

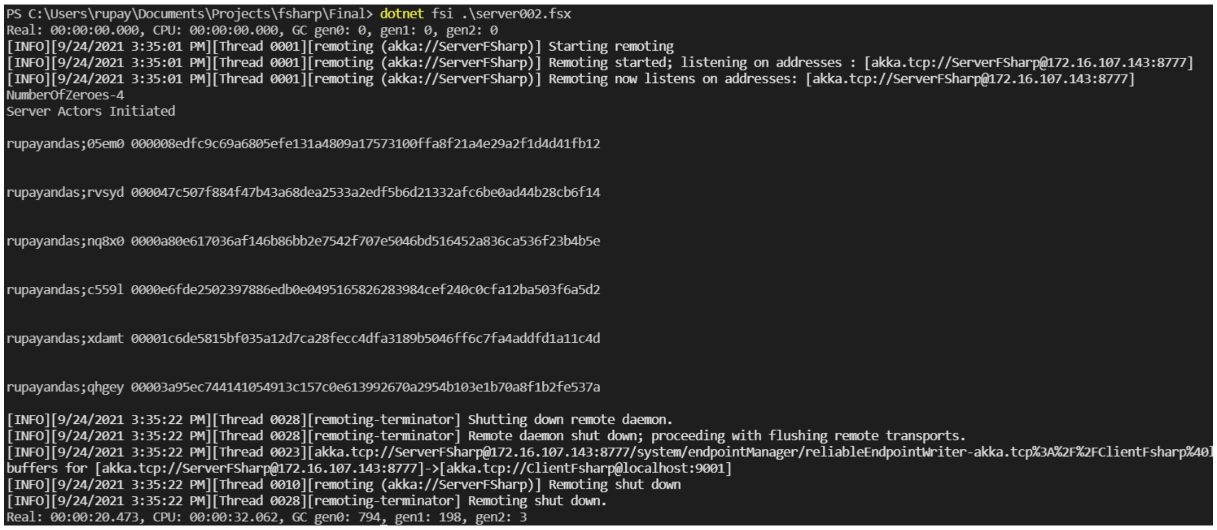
After much investigation we realized that our algorithm can be guessing the same values again and again which is not an intuitive outcome given the sample size available for the random string generator, but seeing that this is the case, we decided to implement a concurrent dictionary to make sure that we don't hash the same string more than once. Knowing that dictionaries have a lookup of O(1) in theory but slightly more in reality and are subject to worsen by the size of the dictionary and noticing that the AKKA often times out halting the algorithm, we decreased the size of the **work unit to only hashing 4 new strings** with SHA256.

Also, we realized that sometimes our worker times out in the process of generating unique strings, so we limited each worker to only generating 6 strings and checking with a concurrent dictionary that the string is unique before relinquishing one chance to hash a unique string. It must be mentioned that strings that have the required number of zeros or more will not be put in the dictionary, so it is possible for our algorithm to find the same string that hashes with the required or more number of leading zeros.

The result of running your program for input 4

Server Output - Coins Found:

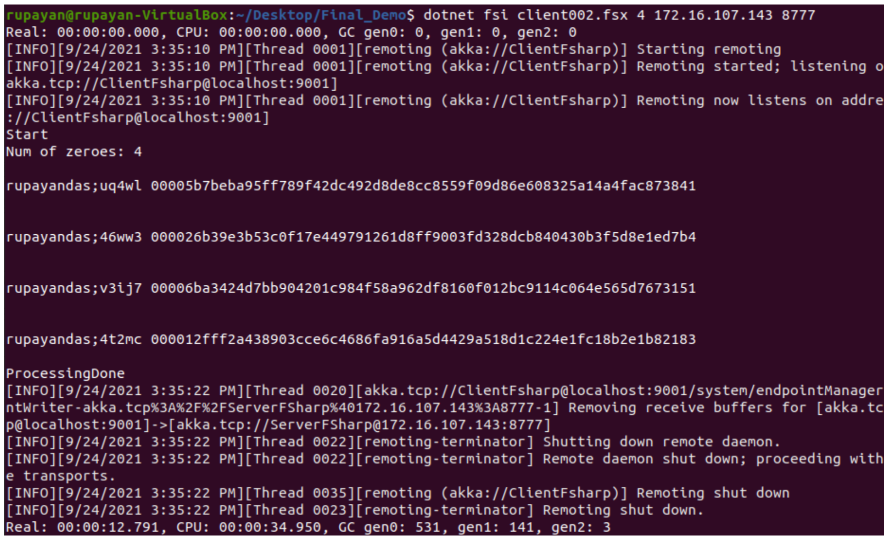
```
rupayandas;05em0     000008edfc9c69a6805efe131a4809a17573100ffa8f21a4e29a2f1d4d41fb12

rupayandas;rvsyd     000047c507f884f47b43a68dea2533a2edf5b6d21332afc6be0ad44b28cb6f14

rupayandas;nq8x0     0000a80e617036af146b86bb2e7542f707e5046bd516452a836ca536f23b4b5e

rupayandas;c559l     0000e6fde2502397886edb0e0495165826283984cef240c0cfa12ba503f6a5d2

rupayandas;xdamt     00001c6de5815bf035a12d7ca28fecc4dfa3189b5046ff6c7fa4addfd1a11c4d

rupayandas;qhgey     00003a95ec744141054913c157c0e613992670a2954b103e1b70a8f1b2fe537a
```



```
PS C:\Users\rupay\Documents\Projects\fsharp\Final> dotnet fsi .\server002.fsx
Real: 00:00:00.000, CPU: 00:00:00.000, GC gen0: 0, gen1: 0, gen2: 0
[INFO][9/24/2021 3:35:01 PM][Thread 0001][remoting (akka://ServerFSharp)] Starting remoting
[INFO][9/24/2021 3:35:01 PM][Thread 0001][remoting (akka://ServerFSharp)] Remoting started; listening on addresses : [akka.tcp://ServerFSharp@172.16.107.143:8777]
[INFO][9/24/2021 3:35:01 PM][Thread 0001][remoting (akka://ServerFSharp)] Remoting now listens on addresses: [akka.tcp://ServerFSharp@172.16.107.143:8777]
NumberOfZeroes-4
Server Actors Initiated

rupayandas;05em0 000008edfc9c69a6805efe131a4809a17573100ffa8f21a4e29a2f1d4d41fb12


rupayandas;rvsyd 000047c507f884f47b43a68dea2533a2edf5b6d21332afc6be0ad44b28cb6f14


rupayandas;nq8x0 0000a80e617036af146b86bb2e7542f707e5046bd516452a836ca536f23b4b5e


rupayandas;c559l 0000e6fde2502397886edb0e0495165826283984cef240c0cfa12ba503f6a5d2


rupayandas;xdamt 00001c6de5815bf035a12d7ca28fecc4dfa3189b5046ff6c7fa4addfd1a11c4d


rupayandas;qhgey 00003a95ec744141054913c157c0e613992670a2954b103e1b70a8f1b2fe537a

[INFO][9/24/2021 3:35:22 PM][Thread 0028][remoting-terminator] Shutting down remote daemon.
[INFO][9/24/2021 3:35:22 PM][Thread 0028][remoting-terminator] Remote daemon shut down; proceeding with flushing remote transports.
[INFO][9/24/2021 3:35:22 PM][Thread 0023][akka.tcp://ServerFSharp@172.16.107.143:8777/system/endpointManager/reliableEndpointWriter-akka.tcp%3A%2F%2FClientFsharp%40l
buffers for [akka.tcp://ServerFSharp@172.16.107.143:8777]->[akka.tcp://ClientFsharp@localhost:9001]
[INFO][9/24/2021 3:35:22 PM][Thread 0010][remoting (akka://ServerFSharp)] Remoting shut down
[INFO][9/24/2021 3:35:22 PM][Thread 0028][remoting-terminator] Remoting shut down.
Real: 00:00:20.473, CPU: 00:00:32.062, GC gen0: 794, gen1: 198, gen2: 3
```

Client Output - Coins Found

rupayandas;uq4wl	00005b7beba95ff789f42dc492d8de8cc8559f09d86e608325a14a4fac873841
rupayandas;46ww3	000026b39e3b53c0f17e449791261d8ff9003fd328dcb840430b3f5d8e1ed7b4
rupayandas;v3ij7	00006ba3424d7bb904201c984f58a962df8160f012bc9114c064e565d7673151
rupayandas;4t2mc	000012fff2a438903cce6c4686fa916a5d4429a518d1c224e1fc18b2e1b82183



```
rupayan@rupayan-VirtualBox:~/Desktop/Final_Demo$ dotnet fsi client002.fsx 4 172.16.107.143 8777
Real: 00:00:00.000, CPU: 00:00:00.000, GC gen0: 0, gen1: 0, gen2: 0
[INFO][9/24/2021 3:35:10 PM][Thread 0001][remoting (akka://ClientFsharp)] Starting remoting
[INFO][9/24/2021 3:35:10 PM][Thread 0001][remoting (akka://ClientFsharp)] Remoting started; listening o
akka.tcp://ClientFsharp@localhost:9001]
[INFO][9/24/2021 3:35:10 PM][Thread 0001][remoting (akka://ClientFsharp)] Remoting now listens on addre
://ClientFsharp@localhost:9001]
Start
Num of zeroes: 4

rupayandas;uq4wl 00005b7beba95ff789f42dc492d8de8cc8559f09d86e608325a14a4fac873841


rupayandas;46ww3 000026b39e3b53c0f17e449791261d8ff9003fd328dcb840430b3f5d8e1ed7b4


rupayandas;v3ij7 00006ba3424d7bb904201c984f58a962df8160f012bc9114c064e565d7673151


rupayandas;4t2mc 000012fff2a438903cce6c4686fa916a5d4429a518d1c224e1fc18b2e1b82183

ProcessingDone
[INFO][9/24/2021 3:35:22 PM][Thread 0020][akka.tcp://ClientFsharp@localhost:9001/system/endpointManager
ntWriter-akka.tcp%3A%2F%2FServerFSharp%40172.16.107.143%3A8777-1] Removing receive buffers for [akka.tc
p@localhost:9001]->[akka.tcp://ServerFSharp@172.16.107.143:8777]
[INFO][9/24/2021 3:35:22 PM][Thread 0022][remoting-terminator] Shutting down remote daemon.
[INFO][9/24/2021 3:35:22 PM][Thread 0022][remoting-terminator] Remote daemon shut down; proceeding with
e transports.
[INFO][9/24/2021 3:35:22 PM][Thread 0035][remoting (akka://ClientFsharp)] Remoting shut down
[INFO][9/24/2021 3:35:22 PM][Thread 0023][remoting-terminator] Remoting shut down.
Real: 00:00:12.791, CPU: 00:00:34.950, GC gen0: 531, gen1: 141, gen2: 3
```

Running time

On server side :

- Real Time - 20.473 seconds (note: this time includes the time required for establishing connection between the server and the client)
- CPU Time - 32.062 seconds
- Ratio - 1.56

On client side:

- Real Time - 12.791 seconds
- CPU Time - 34.950 seconds
- Ratio - 2.73

The coin with the most 0s you managed to find

```
rupayandas;qbqm353kpf    0000000021175debdc235b8e1a7ca70b29edd1f70016348b80622d9dec4ae807
```

The largest number of working machines you were able to run your code with

The largest number of working machines we were able to run your code with was two. We ran the code on two machines. One machine, equipped with a CPU with six cores, ran the server code. The other machine, equipped with a CPU with four cores, ran the client code. The two machines communicate over ethernet on a local area network.

The code can work such that multiple servers can be connected to a single client. And then each server will generate it’s own bitcoin hashes.