

Distributed Operating System Principles (COP5615): Project - 1

Submitted By:

Alekhya Gollamudi, UFID: 5194-2114

Santosh Maturi, UFID: 4533-9141

Problem Definition:

The technique of converting any given key or string of characters into another value is known as hashing. This is generally represented by a shorter, fixed-length value or key that reflects the original string and makes it simpler to locate or use it. In the case of cryptocurrencies like bitcoin, transactions are accepted as input and passed through a hashing algorithm (Bitcoin uses SHA-256) that produces an output of a defined length. In the instance of SHA-256, the output will always be a fixed 256-bits length, no matter how big or little your input is. This is especially important when dealing with large amounts of data and transactions.

By far, the most commonly utilized crypto-currency is Bitcoin. Bitcoins depend on cryptographic hashing to ensure a limited "supply" of coins. The essential component of a bitcoin is an input that produces an output that is less than the required value when "hashed." In reality, the comparison values contain leading 0's. Thus bitcoin must have a specific number of them (to ensure three leading 0's, look for hashes less than 0x001000... or less than or equal to 0x000ff... The hash we are using is SHA-256.

This first project aims to develop a decent solution to this problem that works well on multi-core computers using F# and the actor model.

Background:

- In this project, you have to employ just the AKKA actor library in F#.
- A model similar to the one presented in class for the issue of adding up a lot of numbers may be employed here, in particular, constructing worker actors that are given a variety of topics to solve and a boss that keeps track of all the problems and handles the job assignment.
- Here we are needed to utilize the SHA256 Hash Function and create hashes.
- If we find hashes that have k leading 0's at the beginning, then a Bitcoin is mined.

System Prerequisites:

- Installed the .NET SDK
- Multicore System
- Installed the F# language server for .NET
- If you're using Visual Studio Code, use the Ionide extension for F Sharp.

Assumptions:

- The hash generated should have “K” leading zeros for a successful match.
- As the number of actors grows, so does the number of possible combinations, resulting in the increased hash generation and a higher chance of finding more Bitcoins.

- **Size of work unit:**

The worker receives a mining request with no upper limit on the number of coins to be mined in this program. For bitcoin mining, we launch $2 \times \text{logproc}$ worker processes/node, where logproc is the number of logical processors available on the computer. We also supervise each worker's process and restart it if it fails to guarantee that all worker processes remain operational.

The work unit is three in size. A single worker (i) produces a random worker (ii) Performs SHA-256 Encoding (iii) checks the encoded string for the number of leading zeros .

- **Running Time:**

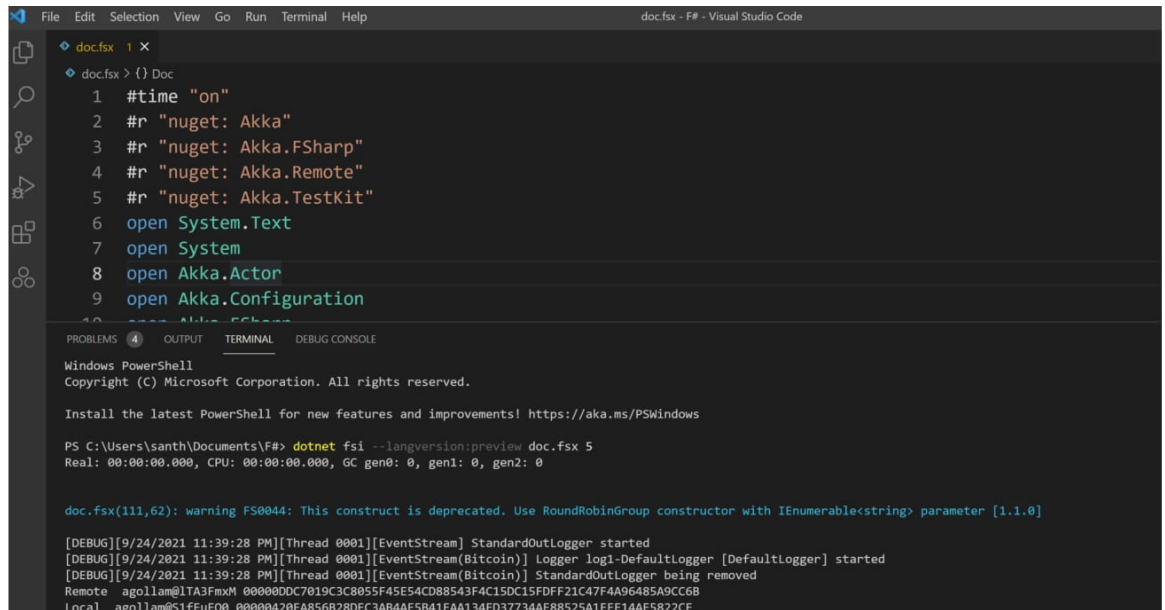
```
Real: 00:02:58.764, CPU: 00:07:41.093, GC gen0: 9734, gen1: 2023, gen2: 9
```

- **CPU/Real-time ratio:**

CPU/real-time ratio would be $(\text{cpu time} / \text{real time}) = 7.41 / 2.58 = 2.87$ (>1 , therefore parallelism exists).

[illegible]

- The coin with the most 0s you managed to find was *5



The screenshot shows the Visual Studio Code interface. The editor window displays a file named `doc.fsx` with the following content:

```
1 #time "on"
2 #r "nuget: Akka"
3 #r "nuget: Akka.FSharp"
4 #r "nuget: Akka.Remote"
5 #r "nuget: Akka.TestKit"
6 open System.Text
7 open System
8 open Akka.Actor
9 open Akka.Configuration
```

The terminal window at the bottom shows the command `dotnet fsi --langversion:preview doc.fsx 5` being executed. The output includes a warning about a deprecated constructor and several debug messages from the Akka framework.

Note: To achieve the solution faster for more than 8 zeros, additional cores and powerful computers are required. I am certain that by connecting my project to several quad-core computers, I will be able to accomplish a larger number of zeros in a shorter amount of time.

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

I only had two machines since I lacked the necessary resources. I am convinced, however, that I can tweak the code to make it function on numerous machines with a single client and multiple servers.

Conclusion:

The CPU/Real-time ratio in the client-server architecture is 2.87, which is higher than the one obtained while running on a single machine for the same amount of computation ($K=4$). As a result, the introduction of a multi-system design has improved performance.