Bitcoin Miner Using Scala and Actor Model

Name: Chen Chen*; UFID: 6667-1010; Gatorlink: alex8937

Version:

1. Scala 2.11.7
2. SBT (the Simple Build Tool) 0.13.9
3. Akka 2.4-SNAPSHOT

Overview:

There are two separate parts of code in the project, server (*boss*) in the ServerBitminer directory and client (worker) in LocalBitminer directory.

On the server-side, the inputs specify the number of leading 0s of the bitcoin and the number of workers (independent of client) working with server. Once having received the inputs, a master actor in the server will distribute the work through a RoundRobinPool and supervise all the workers until one of them finds the required bitcoin. Meanwhile, the server also listens on its hosting IP address for any remote client to join if available. Once one of the workers, no matter server or client side, finds a required bitcoin, the message is sent to the master and every actor terminates.

On the client-side, it starts a local worker to look up the master actor on the server-side by its hosting IP address and participate into mining.

Running the code:

Follow these steps to run the code:

1. `cd` into the Serverminer directory.
2. Type `sbt` to start the interative mode
3. Type `run <num of 0s> <num of workers>` to start the server

Then in a separate terminal window or another computer:

1. `cd` into the LocalBitminer directory.
2. Type `sbt` to start the interative mode
3. Type `run` to start the client
4. `>Enter your target IP: <host IP>`

Structure:

The basic structure of both directories are shown as follows.

```
ServerBitminer
├── build.sbt
├── lib
├── project
│   ├── build.properties
│   ├── Build.scala
│   └── target
├── src
│   ├── main
│   │   ├── resources
│   │   │   ├── application.conf
│   │   └── scala
│   │       ├── Pi.scala
│   ├── target
│   └── test
│       └── scala
└── target
```

```
LocalBitminer
├── build.sbt
├── project
│   ├── build.properties
│   └── target
├── src
│   └── main
│       ├── resources
│       │   ├── application.conf
│       │   └── application.conf~
│       └── scala
│           └── Client.scala
└── target
```

Discussion:

I. Size of work unit

Let me start with describing my algorithm of finding the required bitcoin. Conceptually, it generates all possible suffix strings in an ordered way: first try all the strings containing one character -- basically it goes through the ASCII printable code chart, 95 trials in total. If the required hash key is not found, it then tries all the string with two characters (95*95 trails)... and so on so forth.

In particular, I construct a mapping that transforms all the suffix (starting from a string containing one character) to the domain of natural number (from 1 to +inf). Then the work is evenly divided into M pieces of sub-work, where M is the total number of workers. Hence, each worker is assigned to deal with 1/M amount of entire work. Here is example to better illustrate this idea. Say we have 3 workers on server side, then worker1 takes the work of {1,4,7,...,3k+1}; worker2 takes the work of {2,5,8,...,2+3k}; worker3 takes the work of {3,6,9,...,3+3k} for k={1,2,3,...}. By following this pattern, the i^{th} worker takes the work subset {x| x=M*k+i}.

Since this algorithm essentially splits the entire natural number set into M subsets, each of which contains same amount of countably infinite elements, the size of work unit for each workers is also countably infinite. All workers involved are fairly competing with each other until one of them wins.

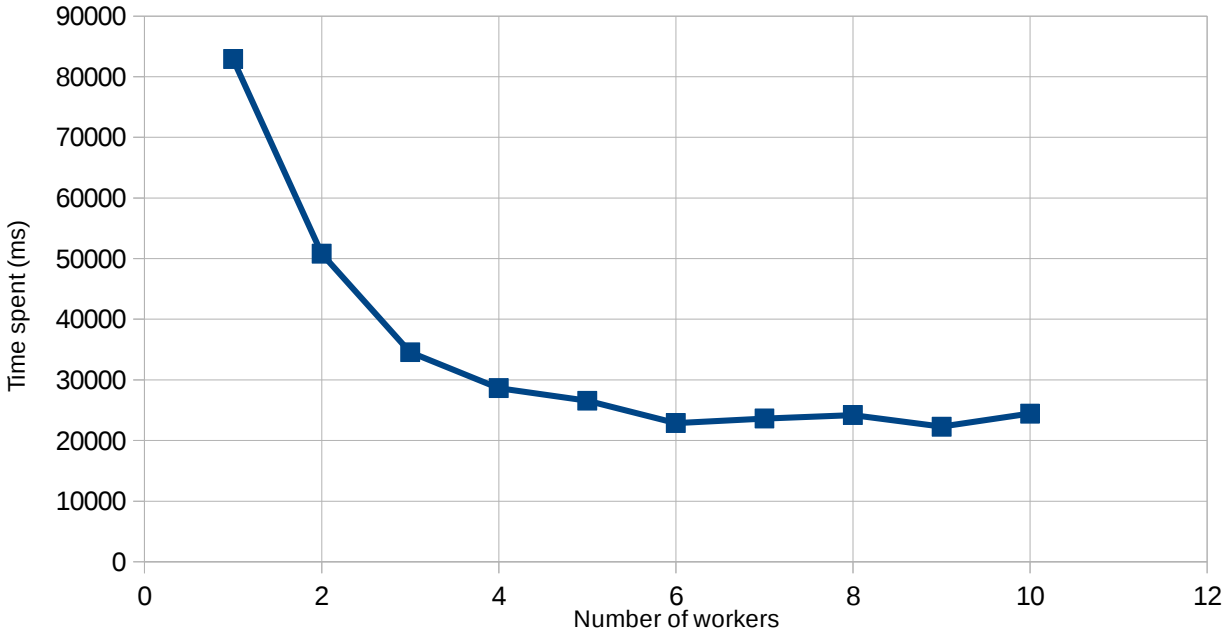
II. Result of four 0s

alex8937;M1$ 00003282d6487cdaa5a4f05efbe3da90f2b3f1ec96b63da1c49254b2766d1125

III. Running time of five 0s

Below is a figure showing the computational time spent for finding a bitcoin starting with five 0s as the number of workers grows. It is observed that beyond around 4 to 5 workers, the efficiency will not keep improving further.

alex8937s@s! 000002d1ca6fde486d7d6d3512d60cb58f239aa738ce4d63967b157d32617eac



IV. Most leading 0s coin found

The coin with most 0s found up to current stage is with eight 0s. The answer is given below.

alex8937}ct=." 00000000760a0c0b2750c098a5529341d318da534606b6bbf0064944f797a8f2

V. Largest number of working machines

Since I was working by myself, I was only able to run my code with one working machine. To ensure that the remote client works proper to connect the server, I test the code as follows:

On the server terminal, a master actor with no worker is established

>run 4 0

Then on the client terminal , it connects the server using the hosting IP address

>run

>Enter your target IP: 127.0.0.1

Then the server terminal displays:

Remote worker found

alex8937nt" 0000857c31f9b90e7f730cee243fd2506c0f94a3f626fb67148b0d9ab508447e