COP5615 – Fall 2015

Alin Dobra

September 3, 2015

- **Due Date:** September 13, Midnight
- One submission per group
- Submit using CANVAS
- **What to include:**
  - `README` file including group members, other requirements specified below
  - `project1.scala` the code for the project

1 Problem definition

Bitcoins (see `http://en.wikipedia.org/wiki/Bitcoin`) are the most popular crypto-currency in common use. At their hart, bitcoins use the hardness of cryptographic hashing (for a reference see `http://en.wikipedia.org/wiki/Cryptographic_hash_function`) to ensure a limited "supply" of coins. In particular, the key component in a bitcoin is an input that, when "hashed" produces an output smaller than a target value. In practice, the comparison values have leading 0's, thus the bitcoin is required to have a given number of leading 0's (to ensure 3 leading 0's, you look for hashes smaller than `0x001000...` or smaller or equal to `0x000ff...`.

The hash you are required to use is SHA-256. You can check your version against this online hasher: `http://www.xorbin.com/tools/sha256-hash-calculator`. For example, when the text "COP5615 is a boring class" is hashed, the value `0xe9a425077e7b492076b5f32f58d5eb6824b1875621e6237f1a2430c6b77e467c` is obtained. For the coins you find, check your answer with this calculator to ensure correctness.

The goal of this first project is to use Scala and the actor model to build a good solution to this problem that runs well on multi-core machines.

2 Requirements

Input: The input provided (as command line to your `project1.scala`) will be k, the required number of 0's of the bitcoin.

Output: Print, on independent lines, the input string and the corresponding SHA256 hash separated by a TAB, for each of the bitcoins you find. Obviously, your SHA256 hash must have the required number of leading 0s ($k = 3$ means 3 0's in the hash notation). An extra requirement, to ensure every group finds different coins, it to have the input string prefixed by the gatorlink ID of one of the team members.

Example 1:

```
scala project1.scala 1
adobra;kjsdfk11 0d402337f95d018438aad6c7dd75ad6e9239d6060444a7a6b26299b261aa9a8b
```

indicates that the coin with 1 leading 0 is `adobra;kjsdfk11` and it is prefixed by the gatorlink ID `adobra`.

Distributed implementation: The more cores you have to more coins you can mine. To this end, enlisting other machines adds to your coin mining capabilities. Extend `project1.scala` so that the argument is a computer address or IP address of the server. This program then becomes a "worker" and contacts the server to get work. This second program will not display anything. All the cons found have to be displayed by the server.

Example 2:

```
scala project1.scala 10.22.13.155
```

will start a worker that contact the scala server hosted at 10.22.13.155 and participates into mining. Hint. when testing this, have your project partner start a server, find the IP address of the server and then start the worker.

Notice, your server should be able to mine coins without any workers but has to accommodate workers as they become available.

Actor modeling: In this project you have to use exclusively the actor facility in Scala (**projects that do not use multiple actors or use any other form of parallelism will receive no credit**). A model similar to the one indicated in class for the problem of adding up a lot of numbers can be used here, in particular define *worker* actors that are given a range of problems to solve and a *boss* that keeps track of all the problems and perform the job assignment.

README file In the README file you have to include the following material:

- Size of the *work unit* that you determined results in best performance for your implementation and an explanation on how you determined it. Size of the work unit refers to the number of sub-problems that a worker gets in a single request from the boss.
- The result of running your program for
  `scala project1.scala 4`

- The running time for the above as reported by time for the above, i.e. run `time scala project1.scala 5` and report the time. The ratio of CPU time to REAL TIME tells you how many cores were effectively used in the computation. If your are close to 1 you have almost no parallelism (points will be subtracted).
- The coin with the most 0s you managed to find.
- The largest number of working machines you were able to run your code with.