**Distributed Operating System Principles (COP5615) : Project1**

**Submitted by:**

**Rohith Kalwa (52075407)**

**Venkata Sai Karthik Metlapalli (65764476)**

**Problem Definition**

The problem is to implement a distributed system using erlang/OTP to generate cryptographic hashes in a concurrent manner using logic similar to bitcoin mining. The task is to generate random strings whose hexadecimal SHA 256 hash digests start with the given number of zeroes as input.

For example, if given input is 6, we need to calculate and output the random string when appended to UFID generates a hash starting with 6 zeros.

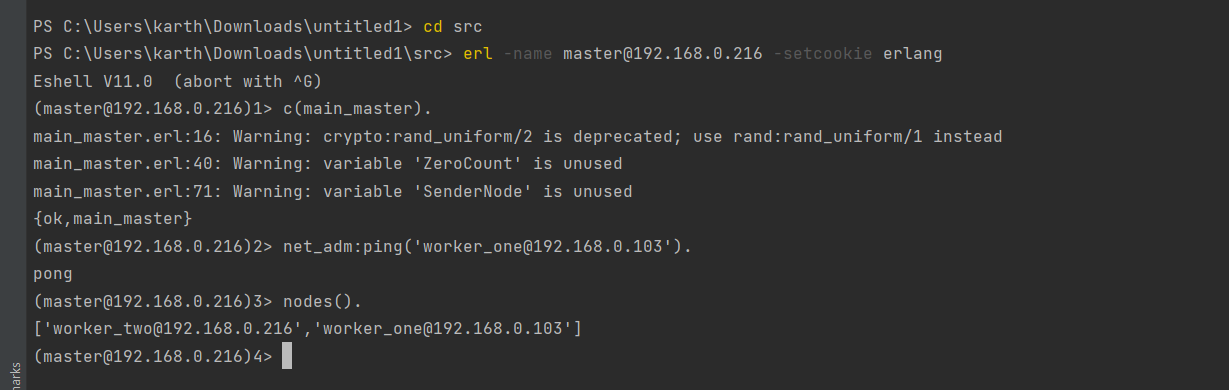
vmetlapalli;RE6Edjuk 000000c44c5139ffa54451f37454d7d0d329cfe2e58a20cf1271fe2a2d678df2

**Implementation**

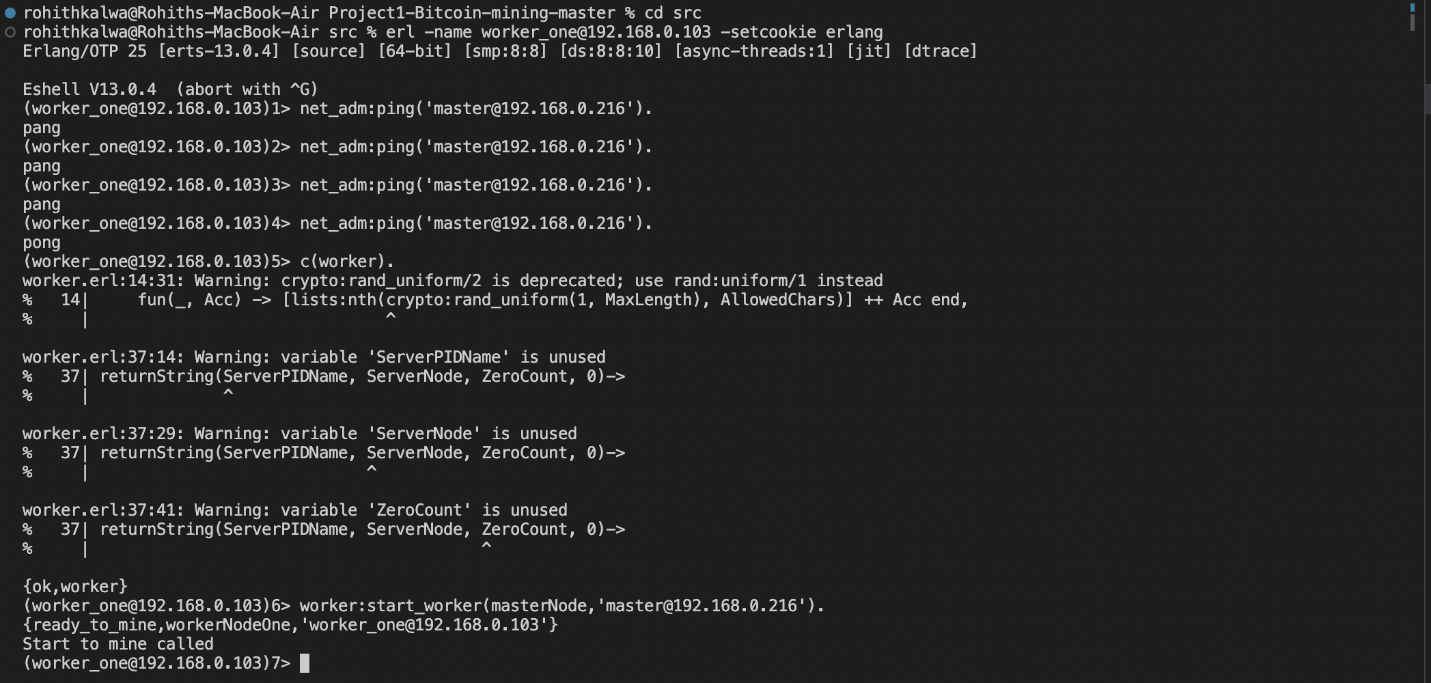
Initially, the master and worker nodes are registered in the same network pool using a shared secure cookie. After this, the worker nodes, who already know the master node and master process name, ping the master node to establish a connection for message passing.

Once the connection is established, first the master process is started followed by the worker processes.

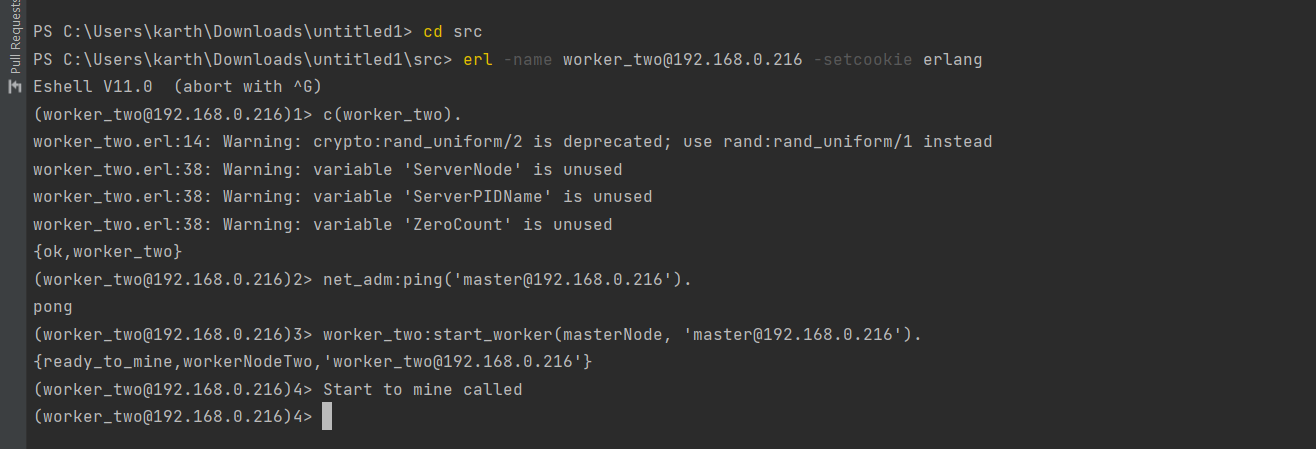
**Master:**



**Worker One( On remote node in a different machine):**



**Worker Two( On different node in same machine):**



Master takes the number of leading zeroes to be present in the bitcoin hash as input and starts mining coins. Once a worker process is available, it sends an available\_to\_mine message to the master. Then the master allocates the mining task to that worker by sending a start to mine message along with the input\_number number of zeroes. And the master listens to the worker and prints output while parallelly mining coins by itself.

**Assumptions**

* All the nodes should be under the same network pool and should be aware of the unique cookie that is used for secure connection.
* The hash generated should have exactly “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 the Bitcoins until a certain threshold.

**Size of the work unit**

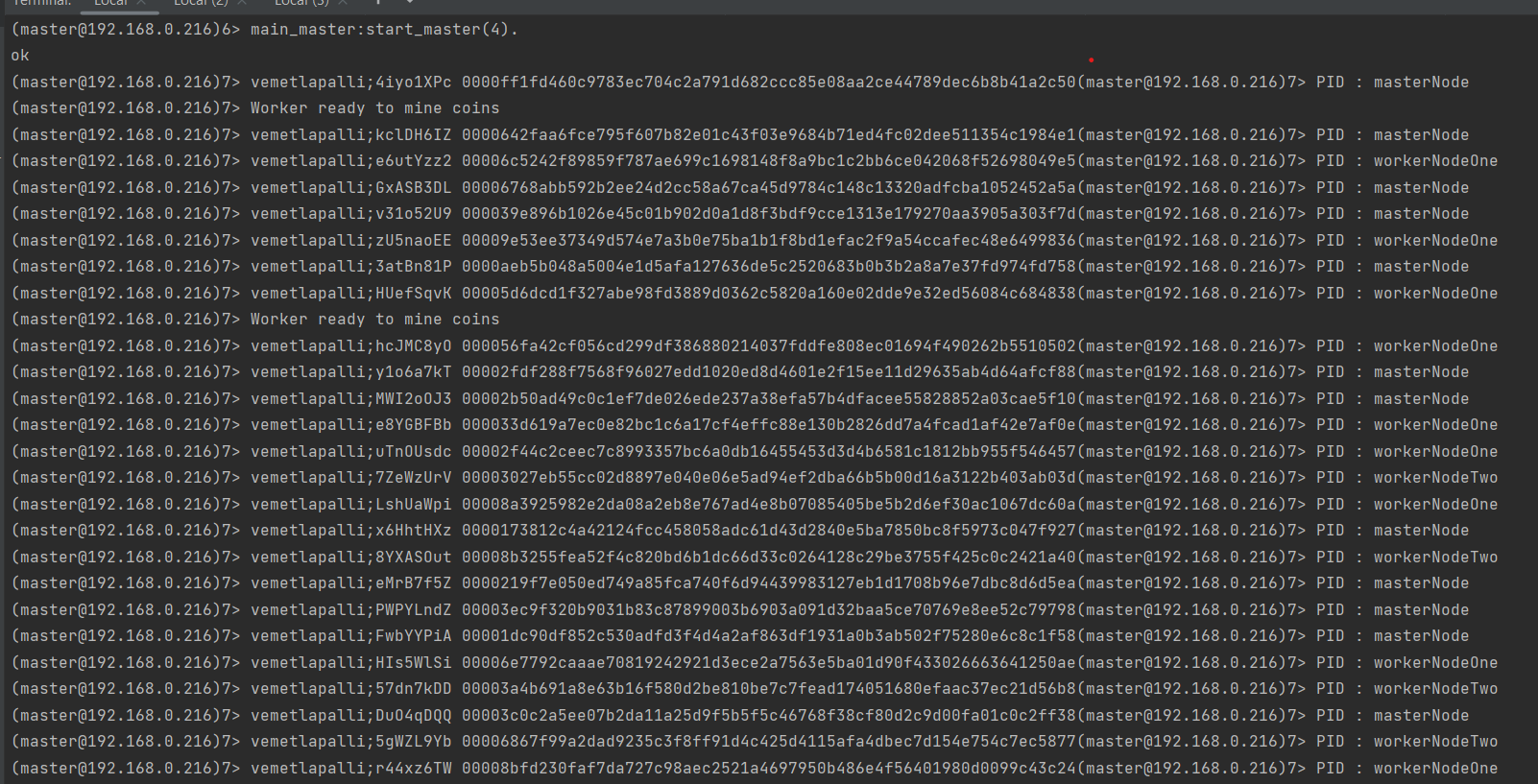
The workers receive a mining request with no upper limit on the number of coins to be mined in this problem. For bitcoin mining we launched 2\*number of logical processors per worker process/node and also in master using **erlang:system\_info(logical\_processors\_available)**

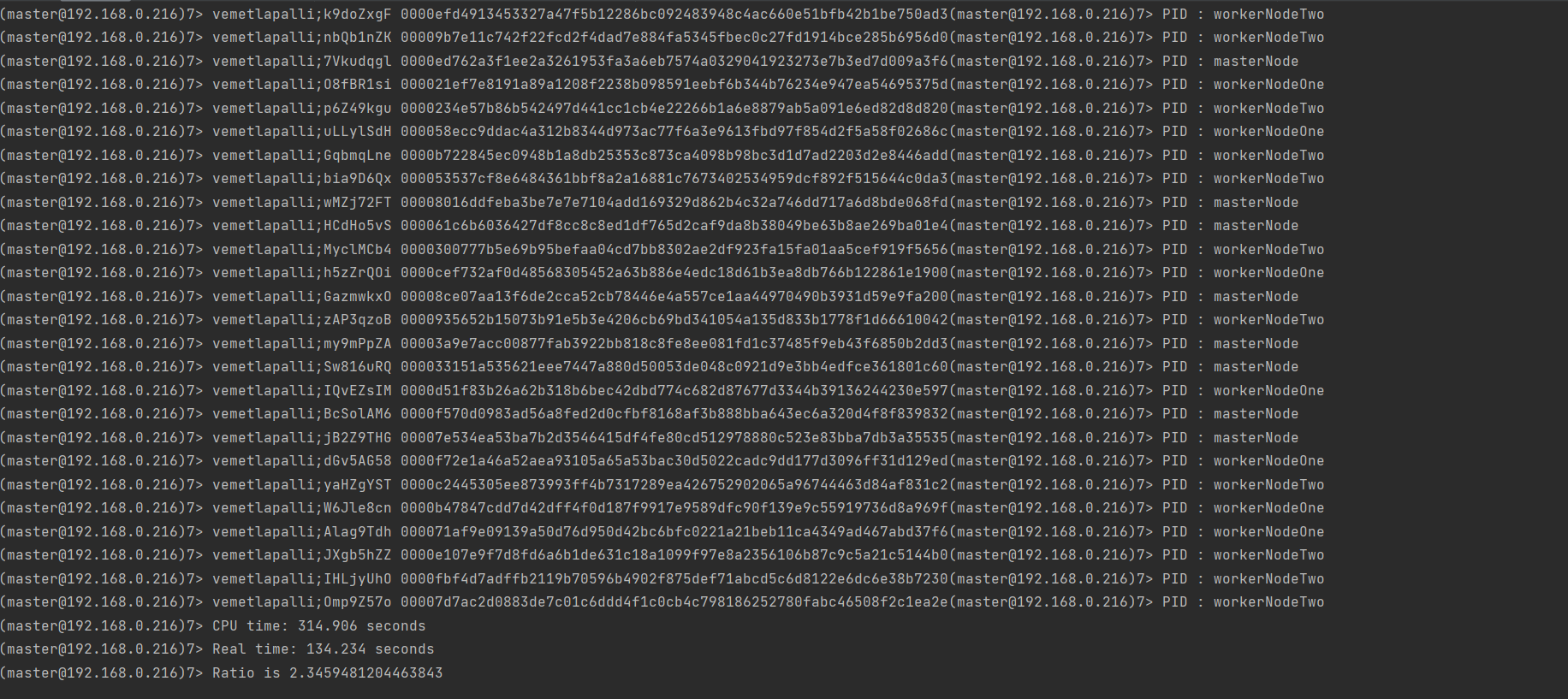
The work unit is 3 in size i.e., each spawned process mines 3 coins before exiting. A single worker (i) Produces/spawns random worker processes, (ii) Each of these performs SHA-256 encoding, (iii) Each of these checks the encoded string for ‘k’ leading number of zeroes and sends it to the master if it meets the criteria.

**The result of running your program for input 4**

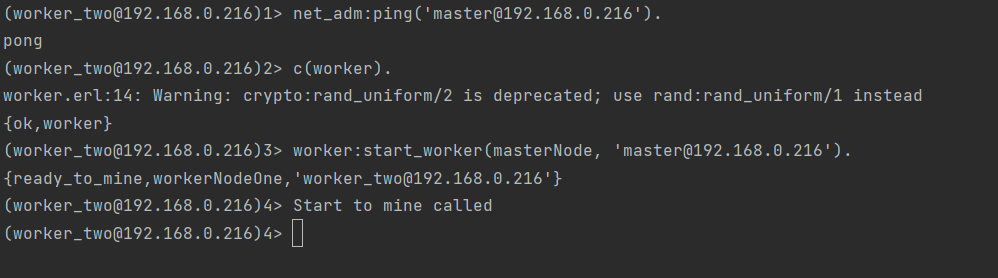
The below are the coins mined that contain 4 leading zeros.

**Master Process Output**

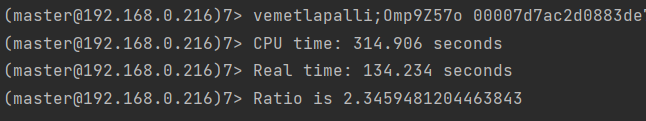


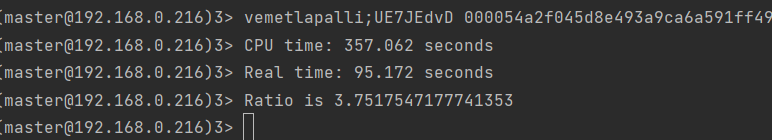


**Worker Process Output**



**The ratio of CPU Time to REAL Time for multiple runs**





CPU/Real-time ratio (i.e., CPU time/Real time) = 357.062/95.172=3.752 (>1, therefore Parallelism exists).

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

The coins with the most 0s we found were 7.

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

Since we only have three machines, we made it work on them. However, we can tweek this code to make it work on numerous machines. The master is capable of connecting with and accepting the output from a large number of workers.

**Conclusion:**

The CPU\_time/Real\_time ratio in the client-server architecture is greater than 2, which is greater than the one obtained while running on a single machine for the same amount of computation. As a result, the introduced multi-system design has improved performance.