Distributed Systems Project 1
BitCoin Mining

Group Details:

Members
1: Manjary Modi ,UFID: 38408368, manjary.modi@ufl.edu
2: Rameshwari Obla Ravikumar,UFID: 16161302, rameshwari.oblar@ufl.edu

Steps to run the application on the Server machine:
1. In shell, navigate to the directory where mix.exs exists
2. Run command

    >mix escript.build
    >./project1 <number of leading 0s>

After step2, the server program will start mining and the generated bitcoins are displayed on the server terminal.
Concurrently, a server node starts that looks out for possible worker connections recursively.

When worker machines are available, they establish connections with the server and the server application automatically creates and schedules mining jobs to that connected worker as when they contact the server. The worker machine performs the mining process and the generated bitcoins are displayed back in the server terminal.

Steps to run the application on the Worker machine:
1. In shell, navigate to the directory where mix.exs exists
2. Run command

    >mix escript.build
    >./project1 <Server Ip Address>

After step2, The worker program will start a node at its end and establishes connection with the server machine. The job gets scheduled instantly and the mining process happens in the background, that is, as soon as the coins are generated it is sent back to the server and nothing gets printed on the worker terminal.

As per the requirement, the mining process happens indefinitely in all the machines. When the server is stopped, the mining process in the worker machines also get stopped.

Note: The workers and the server machines should be connected to the same LAN for the application to work as per the requirement.

Explanation for each of the functions used in the application

1. Main

Used to run the code via escript.build. If the argument is an IP address, it performs worker functionalities else performs server functionalities

2. startServerNode

->Spawns a process to start the server node

->Concurrently, spawns multiple processes to mine bitcoins in the server machine

3. startNode

Starts the server node

4. waitForConnection

Recursively looks up for connected workers

5. distributeWork

Distributes mining jobs to the worker node

6. listen

Stops worker jobs when the server node goes down. This is needed to ensure that workers stop computing as soon as there is no interface available to send the results.

7. startClientNode

Starts the Worker node.

8. startConnection

Used by worker node to establish connection with the server.

9. serverMining

Random input strings of a fixed length are generated and their hashed outputs are checked. If the output contains the required number of leading zeroes, the hashed output is displayed to the server terminal. The whole mining process happens indefinitely.

10. workerMining

Random input strings of random length (different from server input string length) are generated and their hashed outputs are checked. If the output contains the required number of leading zeroes, the hashed output is sent back to the server terminal. The whole mining process happens indefinitely.

<u>Requirement Specifications</u>

1. Work Unit:

As per the document, a work unit is the number of sub-problems that a worker gets in a single request from the boss.

That is, number of jobs that a worker gets in a single request from the sever.

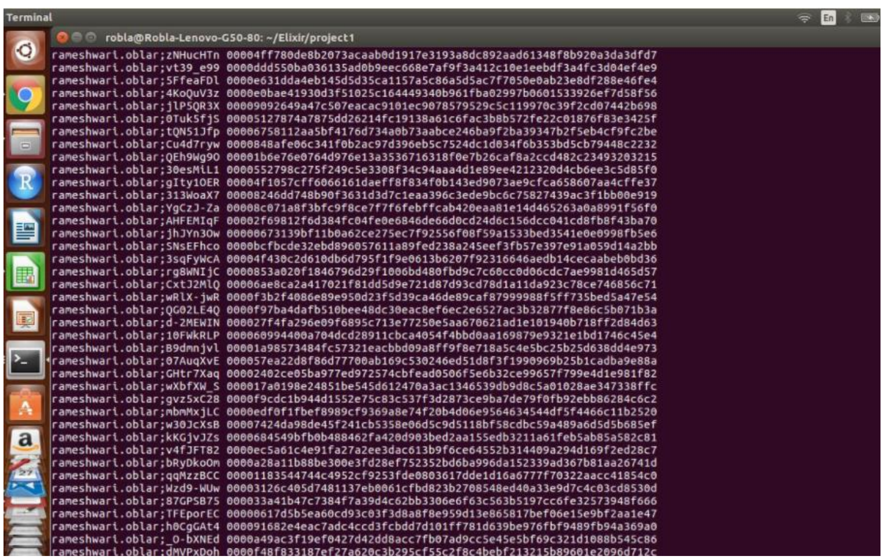
For our application to get the optimal performance, the server has to spawn atleast as many jobs as the number of cores a worker machine has.

->We tested with a cluster of 3 quadcore machines, one acting as a server and the other 2 acting as worker nodes.

->We varied the number of jobs distributed to each of the two worker machines and noticed the CPU utilizations. When the number of jobs were greater than equal to the number of cores in the machine, the CPUs were fully utilized.

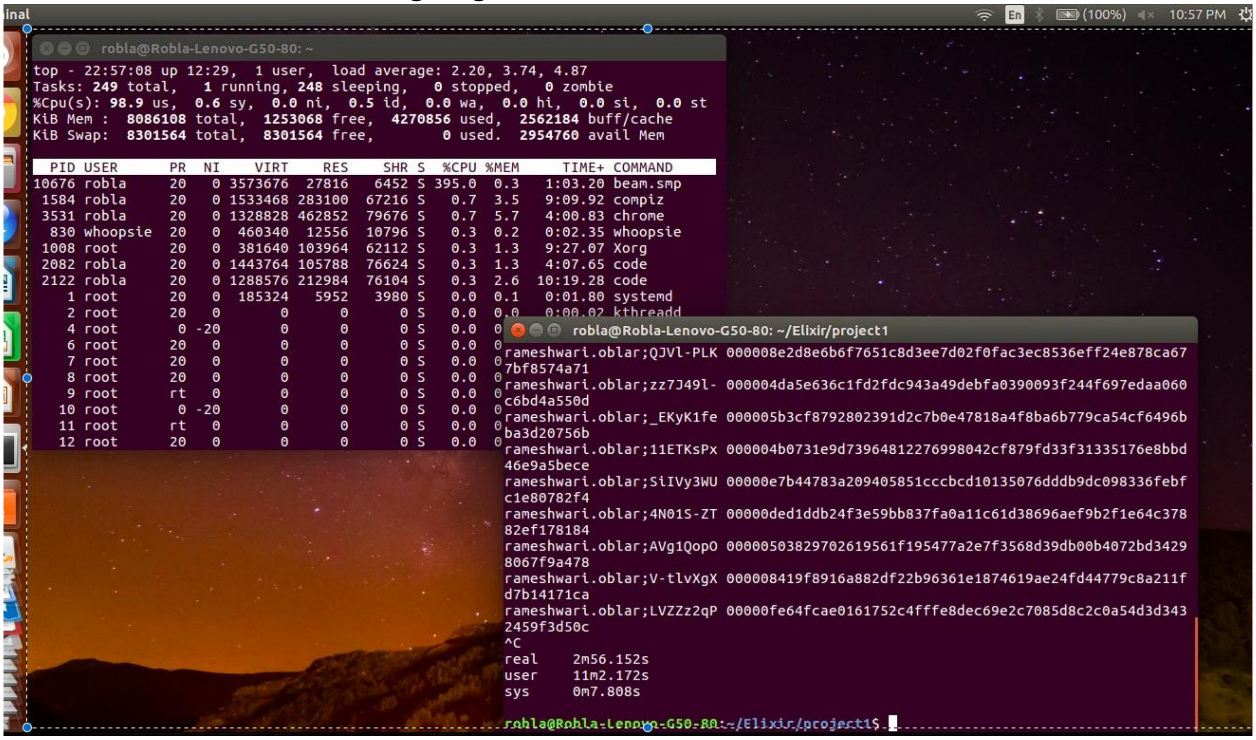
#ServerProcess	#WorkerProcesses	RealTime	UserTime	SystemTime	CPU time(U+S)	Ratio
4	4	0m32.948s	2m2.132s	0m1.932s	124.064s	3.765
8	8	1m17.893s	5m0.172s	0m2.176s	302.348s	3.88
1	4	1m5.792s	2m10.600s	0m0.748s	131.348s	1.996

2. Result for running the program for ./project1 4
rameshwari.oblar;b2C1nQ-6
00000361bc5be79a9eb3cfb91971e6547f1e7aef588709dd5933d7e1e61a249e
rameshwari.oblar;As5aPcWP
0000a4aaf4928942adaf5cb547f9c62504d7a0d5a50d5a0727db64c6ef174181
rameshwari.oblar;TZznGExM
0000c56f3db91d40ff1ab1f8bb348c11d9318d0262faa818a069a506174ef6c3
rameshwari.oblar;Y0NdY9fA
0000399bd6d6c5f3532b1a1e771df811e45f45b151bec159cf545c067eacd375
rameshwari.oblar;nttoNDcZ
000085fb98e9287f72aef08f1280d15c7a3eb6ac1f7dd2037fd76878bef9938c
rameshwari.oblar;c97F2kCg
0000d135166a1b01faa8ba513ba5183c01efa9e887690c183aba89b6fa53fe2d
rameshwari.oblar;-m5mXw2b
00004d287359d8e2579b48d5683d9ad194f02c7df799d63cb0ed45156b28092f
rameshwari.oblar;CBK79XCb
00008db3ee4a23ecac708ff29b74870371d1918714aa910e3d18e10a388615bf
rameshwari.oblar;rcXfpl_V
0000ab127029adb6f1a4132543b1daa65dbde12b006f3ba7ab8a9dfb064a4f6f
rameshwari.oblar;C9KyvV4x
0000f4164c8ed7c1c7534c8b6a5d3d910b68b6b2a095eadb277f50c3ce345341
rameshwari.oblar;LNwukvX3
0000d9d6e8ad9acab5e20b5c370d9254f01a4ab9454a321403b32a1a360bcc57



3. Running time for ./project 5

Below is the sreenshot for string length 5 and CPU time to Real Time = 3.795



4. The coin with the most 0s we managed to find is 8

rameshwari.oblar;y_H_R4Gm

00000000abc35575e405b9b2c79c4de458137867805fc28ba7dee24be802e993

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

3 quadcore machine