Distributed Systems

Bitcoin Mining

Project Execution Instructions:

For Server:

mix escript.build

./project1 k

- Where k represents the number of 0's in the bitcoin

For Worker/Client

./project1 server_ip_address

- Where server_ip_address represents the IP address of server

Implementation Details:

1. **Work Unit:** The nonce generation process for our Bitcoin Mining project should be such that
   - It should not be completely dependent on any randomness function
   - The work must be divided equally among all the miner workers
   - It should not have an upper bound and scalable to any number of workers, which can utilize it simultaneously without exhausting the range
   - It should assign unique work to all the workers

Keeping above points in mind we followed an incremental approach for nonce generation.

The strings are generated by encoding (with base 64) a string integer series starting at "1", which gets incremented by 1 in every iteration.

Number	**Encoding**	**Nonce Generated**
"1"	Base.encode64("1")	MQ==
"2"	Base.encode64("2")	Mg==
"3"	Base.encode64("3")	Mw==
.	.	.
.	.	.
	.	.
"1000000"	Base.encode64("1000000")	MTAwMDAwMA

We have created an assign_work module which assigns work to server as well as client workers as a range of work unit. Each work unit range consists of **1000000** string integers. Workers receive a range of integers and encode each number in the range to base 64 to generate string. Once a worker completes a unit of work, it asks for more work from the server and our module assigns the next available range to the worker.

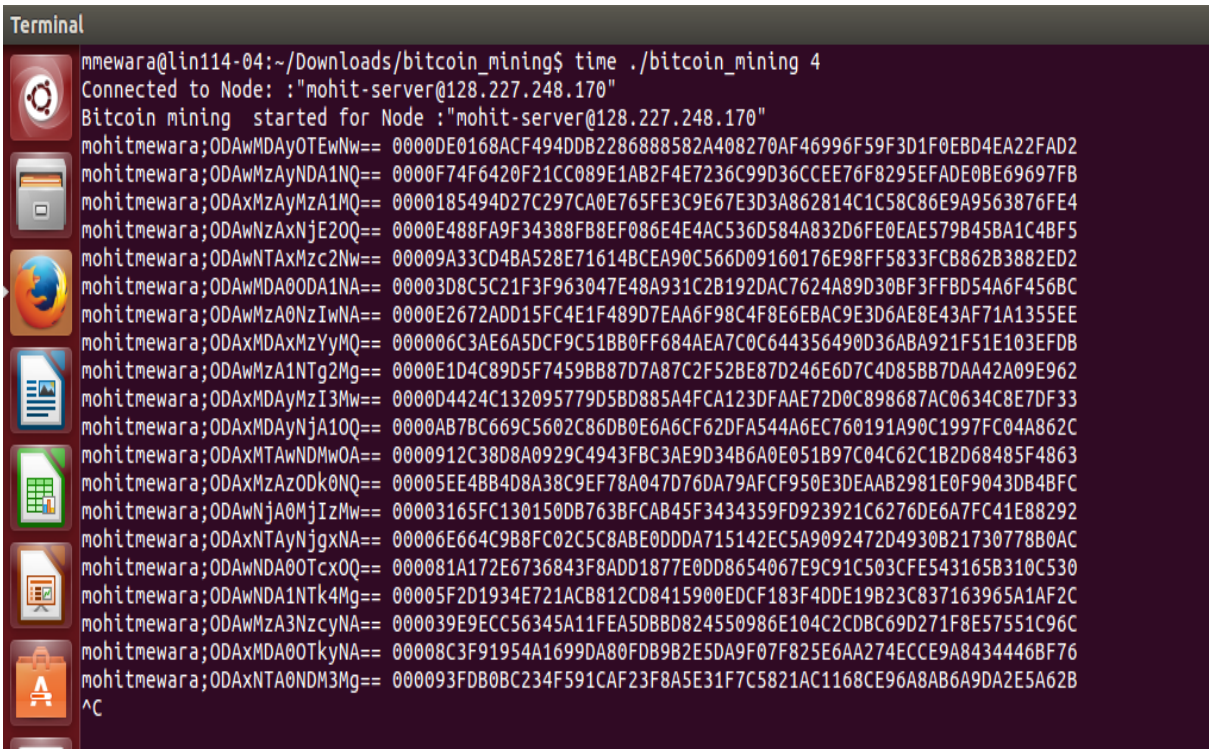
The worker takes around 20 – 30 seconds to use the range (work unit) of 1000000 (million) numbers. If we reduce this range, then the workers flood the message queue of server too frequently and till the time server does not respond back to worker, the worker remain in idle state. With a range of million numbers, we got the average performance of 700% CPU utilization for the 8-core machine. We could easily keep track of the work being executed by different workers and the workers does not flood message queue too frequently of server.

Workers	**Range**
Server	1-1000000
Worker1	1000001-2000000
Worker2	2000001-3000000
.	.
.	.
Worker1 asks for rework	5000000 - 6000000
Worker2 asks for rework	6000000 - 7000000
.	.

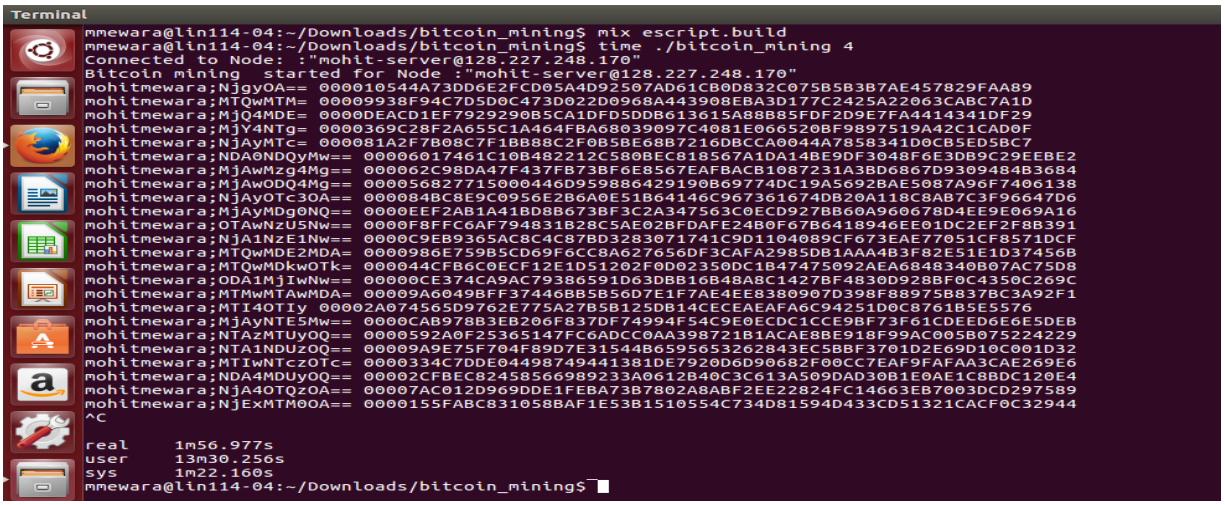
The benefits of the above approach are:

- Since multiple strings can never generate same encoded string, our nonce will remain **unique for every worker** and hence there will never be any duplicate work.
- We are diving the range equally, hence the task is being **equally divided among workers**
- It makes our model **scalable**, as there is no upper bound on range of numbers and any number of workers can connect the server and ask for work.

2. The **result** of running the program for 4 0's bitcoin:
   Please find below the screenshot:



3. We ran our program 'bitcoin_mining' on an **Intel i7 machine with 8 cores** in UF Computer Science Lab (Dungeon Lab) and recorded the below timings:

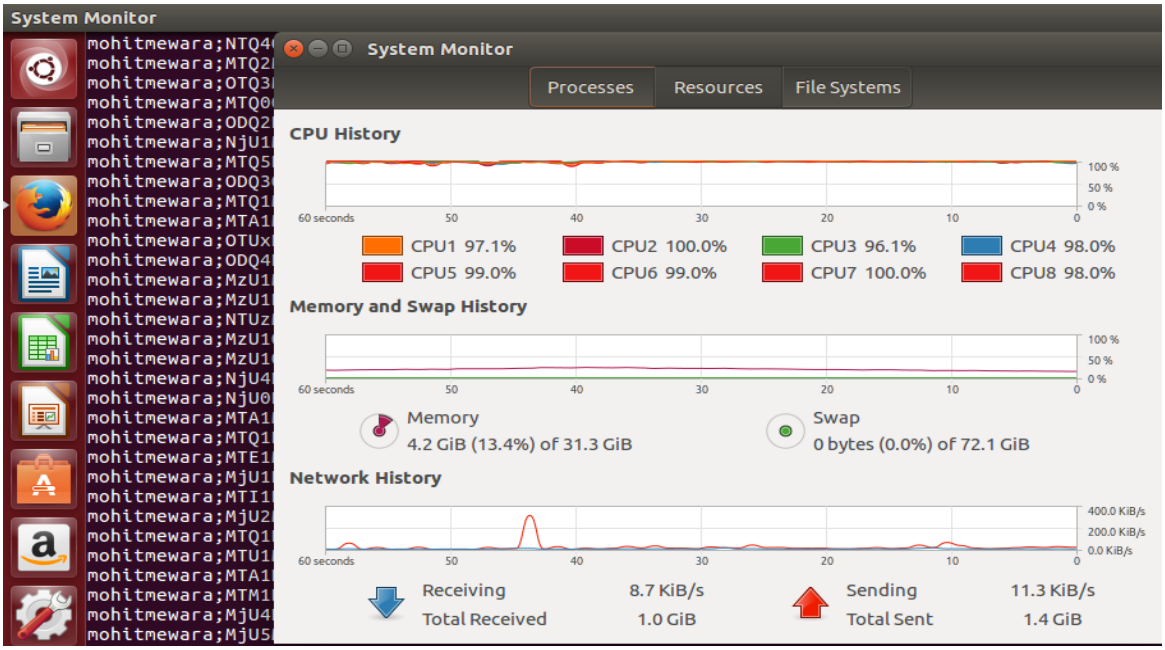


Real : 1m56.977sec = 60+56 ➔ 116 seconds
User : 13m30.256sec = 60+56 ➔ 810 seconds
Sys : 1m22.160sec = 60+56 ➔ 82 seconds

Ratio of CPU time to Real time = User/Real = 810/116 = **6.98 ~ 7**

Hence our project is giving a performance of **700%.**

The below is the screenshot of CPU utilization of all the cores. Almost all the cores are working on 100% CPU utilization.



4. We found the below coin with most number of zeroes:

mohitmewara;MTMzMDk4MTUyOA==
000000052A1EA17E076695CBA51763D85205DA99816B32EC98500861E079A0BC
Number of zeroes in our Bitcoin: **7**

5. The largest number of working machines we tested our code, was with 4 machines where 1 machine acted as a server and other three workers acted as clients.