Horizontal Scaling for an Embarrassingly Parallel Task: Blockchain Proof-of-Work in the Cloud

Unit Code: COMSM0010
Aaron Wray
(aw16997)

INTRODUCTION

This report outlines the Cloud Nonce Discovery (CND) system, which utilizes horizontal scaling hosted by Amazon Web Service's (AWS) Elastic Compute Cloud (EC2) for a parallel computational task. The computational task is to compute the proof-of-work for a Blockchain. The report will describe CND by discussing its design and performance.

1 BACKGROUND

In a Blockchain distributed ledger protocol, the proof-of-work stage consists of using a block of data and an arbitrary random 32-bit number that is only used once (nonce), as inputs to the SHA256 cryptographic hash function. The output of SHA256 is then used as input to the same function again. The output of this is a hash value, a random number in the range 0 to $(2^{256} - 1)$. The aim of proof of work is to determine whether a nonce is golden. For a given data block, a nonce is golden if the hash value returned has a difficulty-level D leading number of bits which are zero.

2 PROBLEM ANALYSIS

The parallel computational task is to brute force nonces used in the proof-of-work until a particular nonce is golden. The task itself has Single Instruction Multiple Data (SIMD) architecture. Multiple nonces can be checked to determine whether they are golden, simultaneously using the CND. This architecture is suitable for parallelism as we can give each worker a different set of inputs.

The CND splits the proof-of-work task amongst N workers in a given time limit T, by calculating the number of nonces a worker can perform per unit of time S. Multiplying S and T gives the total number of nonces that can be checked within a given time limit by an individual worker x. x is used to calculate a range of nonce values for each worker to brute force. For example, with 2 workers the range of nonces checked by worker 1 is 0 to $x - 1$ and for worker 2 the range is x to $2x - 1$. Therefore, using N workers Nx nonces can be brute forced in a given time T.

Experiment 1 is used to determine S for each type of worker available. In *Experiment 1*, 100 trials were performed for checking 100,000 nonces after using the CND, with D set to 256. The CND was used on 2 different workers available: an Intel Core i5 processor and AWS t2.micro instance (Intel Xeon Core). The results from *Experiment 1* were used to calculate the average runtime for checking 100,000 nonces shown in Table 1.

	Average Runtime (seconds)
Intel Core i5 Processor	0.633
AWS t2.micro instance (Intel Xeon Core)	0.619

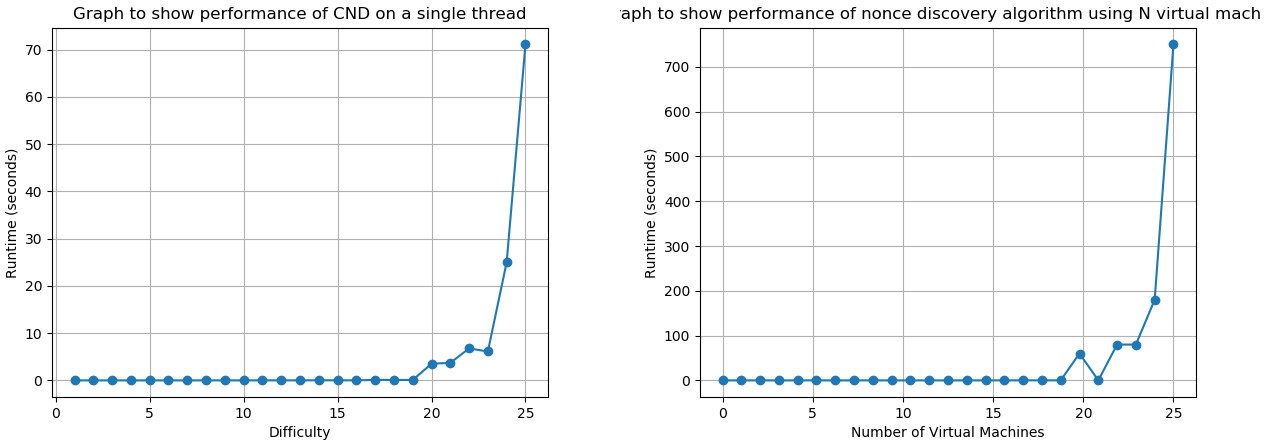
Table 1. Displays the average runtime after 100 trials, for checking 100,000 nonces after using the CND.

The average runtimes were used to calculate the number of nonces that can be checked per unit time by each worker available. S for the Intel core i5 processor and the t2 micro instance were determined to be 157,998 and 161,551 respectively.

3 RESULTS

3.1 Sequentially

Experiment 2a and *Experiment 2b* were ran to test the performance of CND on the available individual workers, without any form of parallelism. For both experiments the runtimes were recorded after running the CND on a single AWS t2.micro instance and a single thread of an Intel Core i5 processor with varying difficulty levels, the results are shown in Figure 1.



(a) A graph that displays the result of running CND on a single thread of an Inter Core i5 processor.

(b) A graph to display the results of running CND on a single Virtual Machine.

Figure 1. Graphs to display the results of *Experiment 2a* and *Experiment 2b*.

The results of the *Experiment 2a* and *Experiment 2b* show that the nonces are discovered almost instantly for difficulty level in the range 0 to 20. The results also indicate that the runtime of CND increases exponentially with difficulty.

3.2 Parallelism using Multiple Threads

CND is able to use multi-threading to split the processing across multiple on-board processors. *Experiment 3* was conducted to show the performance of CND using multiple threads on an Intel core i5 processor. The maximum number of onboard processors that were available were 4 and the results the ex-

periment are displayed in Figure 2.

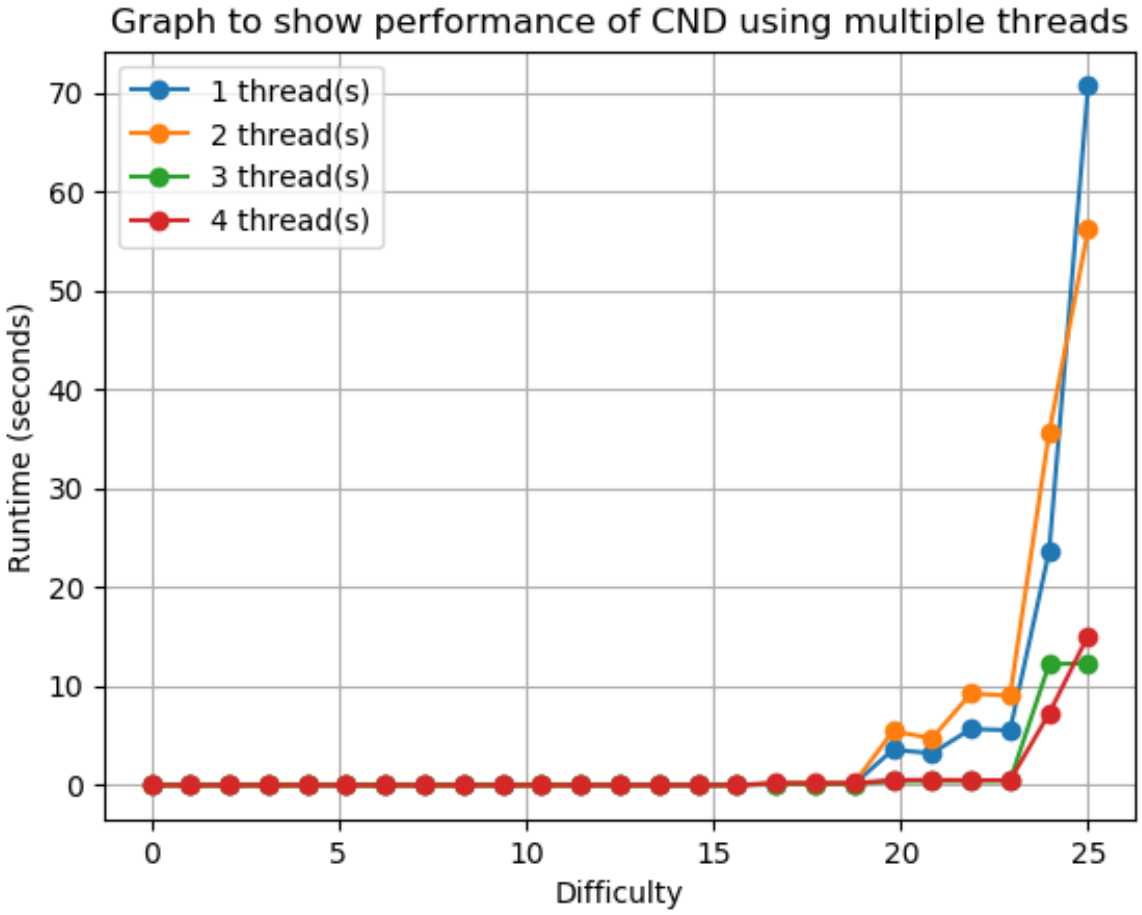


Figure 2. Displays the runtime of CND using multiple threads on an Intel Core i5 processor.

In general using more threads improves performance of CND. However, CND using 1 thread is faster than CND using 2 threads for the first 24 difficulty levels, this suggests that the golden nonce is in the range 0 to $x - 1$ for these difficulty levels. Similarly, CND using 3 threads for the maximum difficulty of 25 was faster than when using 4 threads, which suggests that the golden nonce is in the range $2x$ to $3x - 1$ at a difficulty level of 25.

3.3 Parallelism in the Cloud

The CND allows a user to specify N the number of Virtual Machines (VMs) to be used in parallel. CND then remotely starts up N VMs and runs the proof-of-work on each of them and reports back as soon as one of the VMS has found the golden nonce. *Experiment 4* ran the CND using up to 16 different virtual machines with the difficulty level set to 24. The VMs used in the experiment are the AWS t2.micro instance. The results of *Experiment 4* are displayed in Figure 3.

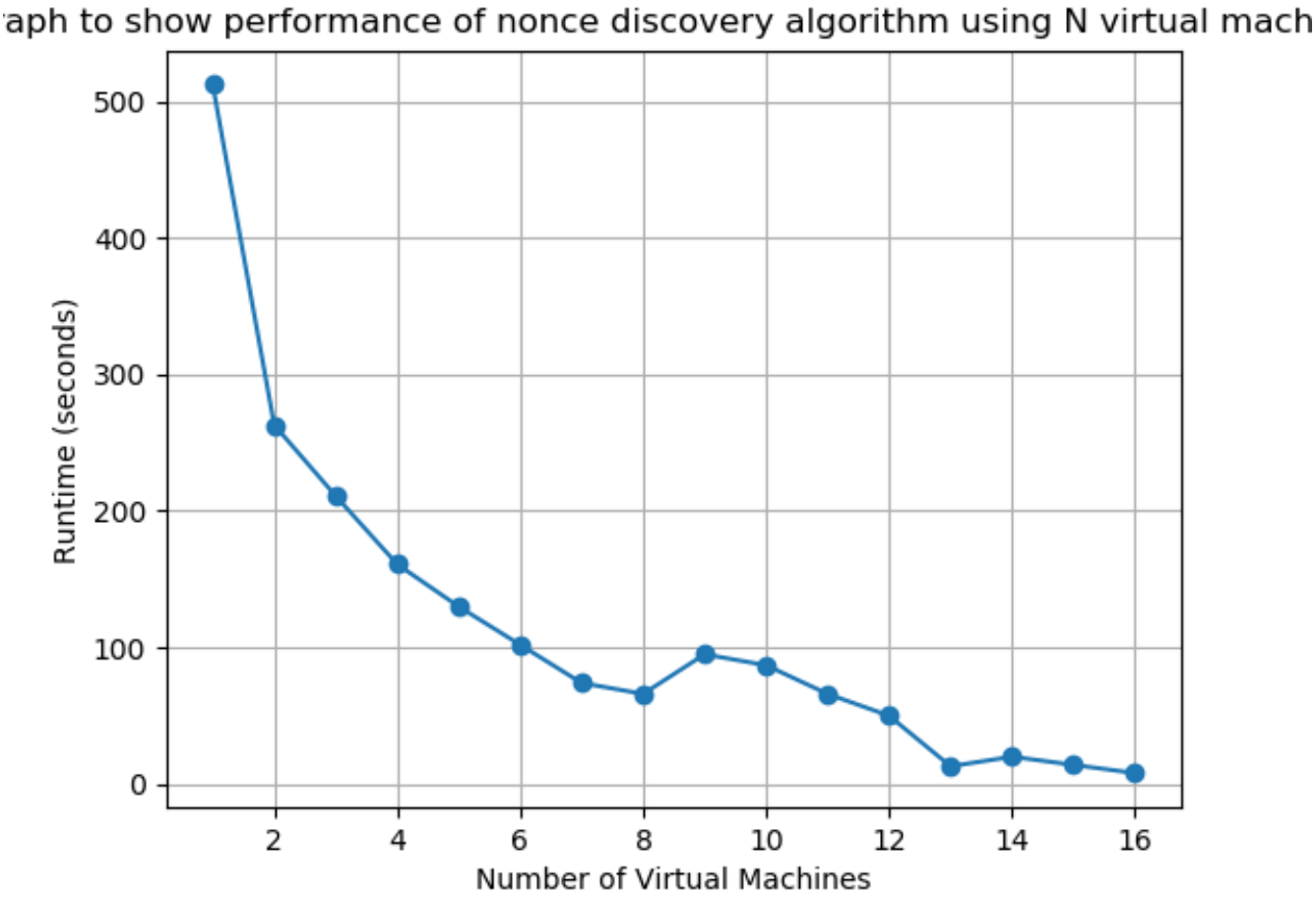


Figure 3. Displays the runtime of CND using multiple Virtual Machines hosted by AWS.

The results of this experiment show that horizontal scaling improves the performance of CND. The relationship between runtime with the number of virtual machines is almost logarithmic.

The final Experiment, *Experiment 5* was ran to determine the performance of CND using 16 VMs with a varying difficulty level. The results of *Experiment 5* are displayed in Figure 4.

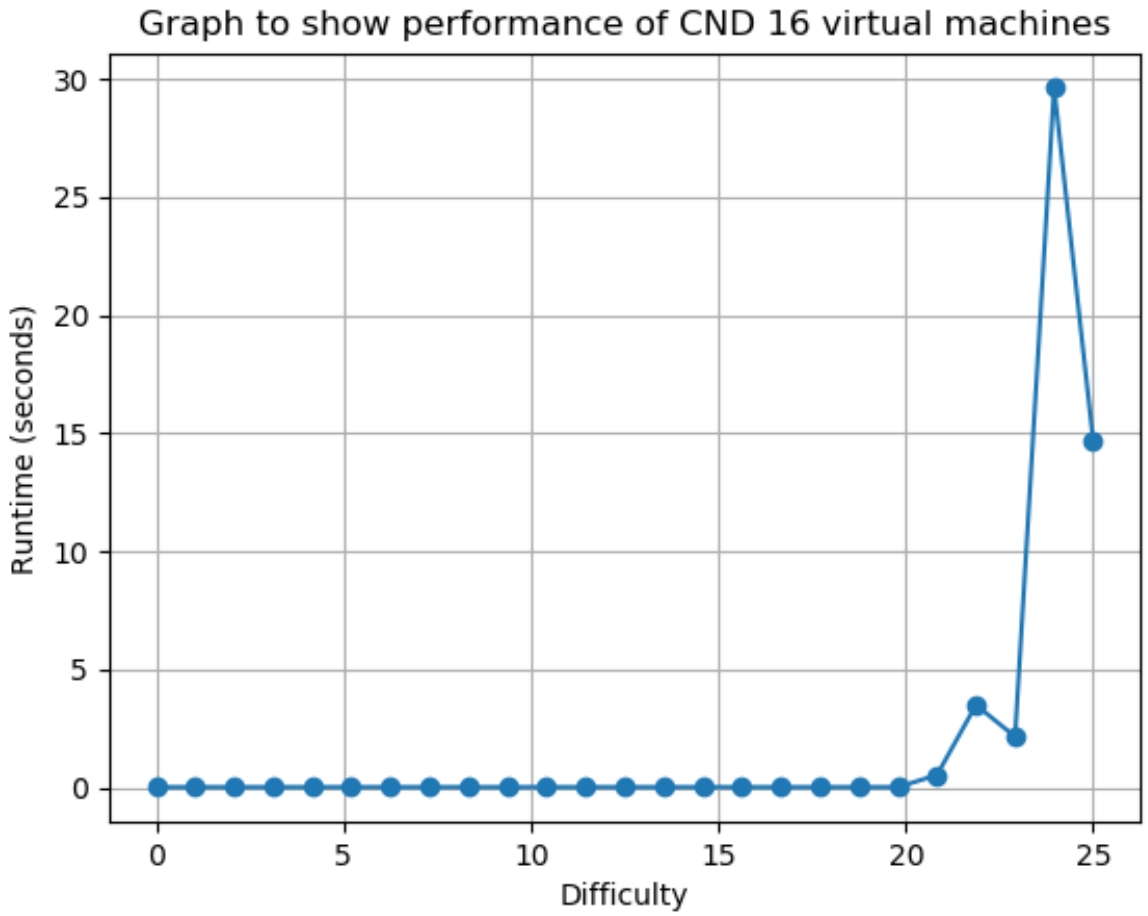


Figure 4. Displays the runtime of CND using 16 Virtual Machines, with difficulty set to 24.

Using 16 Virtual machines at once significantly reduces the runtime of CND. The results from *Experiment 5* show that using horizontal scaling with multiple virtual machines in this way gives the greatest performance compared with any other method the CND has to offer.

4 AUTOMATION

The CND can automatically determine a desired amount of virtual machines N to use to find a golden nonce in a given amount of time T and with a confidence L between 0 and 1.

The SHA256 hash function hashes to 2^{256} different values. The total number of golden nonces that can be found are 2^{256-D}, where D is the difficulty for the proof of work. The probability p that a nonce produces a golden hash value is expressed in equation 1 and 2.

$$p = \frac{2^{256-D}}{2^{256}} \qquad \textbf{Equation } 1.$$

$$p = 2^{-D} \quad \textbf{Equation } 2.$$

The success of each trial is based on whether a nonce is golden or not. Each trial to determine whether a nonce golden is random and independent. Therefore, the discrete random variable can be modelled as a binomial distribution, where p is the probability that a nonce produces a golden hash value.

The expectation E of a binomial distribution can be calculate using equation 3. The expected value when running the CND should be 1, as the system should find at least 1 golden nonce. Therefore, the number of trials that need to be performed by the CND n until at least 1 golden nonce is found is 2^D shown in equation 4, 5 and 6.

$$E = np \quad \textbf{Equation } 3.$$

$$1 = np \quad \textbf{Equation } 4.$$

$$n = \frac{1}{2^{-D}} \quad \textbf{Equation } 5.$$

$$n = 2^D \quad \textbf{Equation } 6.$$

Using the gathered performance statistics from *Experiment 1.* The total number checks performed x by a single virtual machine in a given time T is already known. Therefore, the desired number of virtual machines N with confidence L can be calculated this is shown in equation 7.

$$N = \left\lceil \frac{n}{x} * L \right\rceil \quad \textbf{Equation } 7.$$

5 CONCLUSION

Cloud technologies provide an almost unlimited resource of compute power, which is useful, for problems involving horizontal scaling as an algorithm can be split over many virtual machines. The CND system is a good example of how theses resources can be used, as the report has shown the CND to improve the performance of a parallel computation task using horizontal scaling.