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

Xiaoyue Xiao (1920991)
Course: Cloud Computing
Date: 6th December 2019
https://github.com/shawvey/Cloud-Nonce-Discovery

I. Introduction

The purpose of this coursework is to create Cloud Nonce Discovery (CND) system. CND system allows users to directly specify a large number of options, such as the numbers of virtual machines (VMs) N, maximum time T and difficult-level D. When users run CND script, CND system will start up N virtual machines and use brute-force search to find a golden nonce within maximum time T. When CND system finds a nonce whose second hash value has D leading zeros after double hashing, this nonce will be regarded as a golden nonce. After that, CND system will print the golden nonce as well as its hash value, returns log files and then immediately shuts down all the VMs.

This report will first introduce the crucial proof of work (PoW) stage of Blockchain and then will pay attention to the Boto3 framework which is a useful Python interface to Amazon Web Services (AWS). According to Figure 1, CND system uses several AWS free-tier services, such as Amazon Elastic Compute Cloud (EC2), Amazon Simple Queue Service (SQS) and Amazon Simple Storage Service (S3). These AWS services will also be introduced in this report. Next, this article will describe a Python library named Fabric which is designed to execute multiple shell commands in remote VMs over Secure Shell (SSH) protocol. The decorator @parallel from Fabric will be used in CND system for an embarrassingly parallel task. The following part is to show the entire process of CND system. Finally, a few drawbacks and future research directions for CND system will be mentioned.

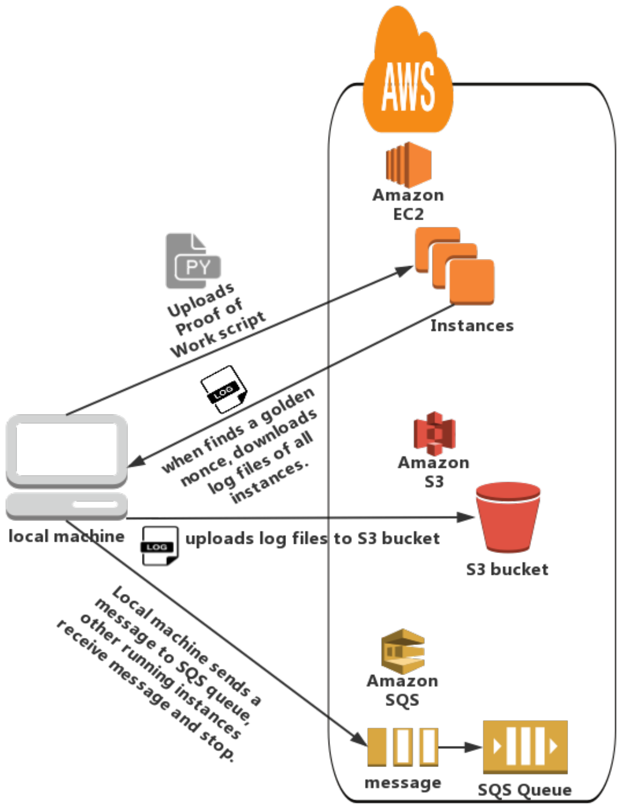

AWS
Amazon EC2
Instances
PY
Uploads Proof of Work script
LOG
when finds a golden nonce, downloads log files of all instances.
Amazon S3
local machine
LOG
uploads log files to S3 bucket
S3 bucket
Local machine sends a message to SQS queue, other running instances receive message and stop.
Amazon SQS
message
SQS Queue

Figure 1: The connection between the local machine and three free-tier AWS services.

II. Proof of work

CND system checks whether a nonce is a golden nonce or not by running proof of work script. In the PoW Python script, a Python function called `goldennonce()` receives two parameters from CND script, which are difficult-level D and a decimal nonce respectively. This function will create a string with consecutive D zeros. Take D value is equal to 5 as an example, the generated string is '00000'. The input nonce is appended to the block whose data is defined to the string 'COMSM0010cloud', and the new string is subjected to a double SHA256 hashing algorithm using Python hashlib module. If the resulting hash value has D or more than D leading zeros, this nonce is referred to as a golden nonce and the details of this golden nonce will be clearly printed.

III. Boto3 framework

Boto3 is an Amazon Web Services (AWS) SDK for Python programming language. Boto3 provides an easy API to use, which allows users to start, stop, and manage all AWS services, such as SQS and EC2 [1].

Developers have to configure authentication credentials before using Boto3. The credentials configuration file should be stored in the *~/.aws* folder. The contents of credentials can be found in the account detail interface of AWS website and they will be updated after session timeout expires. This credentials file enables Boto3 to connect a specific AWS account.

IV. Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) is an important service that enables developers to create instances and run them on remote Amazon data centers [2]. Users can choose suitable configurations for their instances from a variety of machine configurations with different performances based on their requirements. As can be seen in Table 1, different types of EC2 instances have different runtimes when they all start up 5 instances and find the same golden nonce with two leading zeros. Users also choose the number of instances that they want to start up. According to Table 2, it is clear that users will find the golden nonce on average of N times faster than using a single C4.xlarge instance if users start up N C4.xlarge instances.

Table 1: Different instances have different performances.

Instance Type	***Runtime***
T2.micro	13.73945 seconds
C4.xlarge	13.80175 seconds
C5.large	13.53883 seconds

Table2: Different numbers of C4.xlarge instances result in different runtimes for finding the same golden nonce.

Number of instances	***Runtime***
1	66.72627 seconds
2	34.27059 seconds
3	21.93457 seconds
4	17.11261 seconds
5	13.80175 seconds
6	10.89037 seconds
7	9.97410 seconds
8	8.81076 seconds

CND system calls `boto3.client()` function to connect EC2 and then system will set a variety of configurations by calling `run_instances()` function showed in Figure 2, such as Amazon Machine Image (AMI), Instance Type, Security Groups and so on.

```
ec2.run_instances(ImageId=ubuntu_id,I
nstanceType=type,KeyName=key_pair,Sec
urityGroups=['SecurityGroupName'],Min
Count=int(N),MaxCount=int(N))
```

Figure 2: Boto3 creates EC2 instances.

The system needs to wait until N instances are running. In order to ensure that all instances are running, the waiting time is set to 100 seconds. According to Figure 3, when all instances get ready, the system will obtain public IP addresses from all VMs and puts them in *MyHosts* array.

```
MyHosts = []
for r in Reservations['Reservations']:
  for instance in r['Instances']:
    MyHosts.append(instance['PublicIpAd
dress'])
```

Figure 3: Store public IP addresses.

There is a `Scram()` function in CND system, which can be seen in Figure 4. When the system runs until maximum time has reached or finds a golden nonce, it will call `Scram()` function to shut down all running VMs. `Scram()` function first picks out those running instances and calls `terminate_instances()` function to shut down them.

```
def Scram():

   ec2=boto3.client('ec2',region_name='
us-east-1')
   filters = [{'Name':'instance-state-
name','Values': ['running']}]
   Reservations=ec2.describe_instances(
Filters=filters)
   for r in
Reservations['Reservations']:
      for instance in r['Instances']:
         n=instance['InstanceId']
         ec2.terminate_instances(Instanc
eIds=[n])
```

Figure 4: Boto3 terminates running EC2 instances.

V. Amazon SQS

Amazon Simple Queue Service (Amazon SQS) is a distributed message queuing service that enables VMs to communicate. Boto3 calls `boto3.client()` function to establish a connection between boto3 and Amazon SQS service. After that, the system calls `create_queue()` function to create a queue named *CloudComputing* [see Figure5].

```
Sqs = boto3.client('sqs')
Response = sqs.create_queue(
      QueueName='CloudComputing',
)
```

Figure 5: Boto3 creates SQS queue.

If one virtual machine finds a golden nonce, this VM will return details of the golden nonce to the system which will be emphasized in the following Fabric part. Figure 6 demonstrates that CND system sends a message to the specific *CloudComputing* queue while it receives messages from one instance.

```
Response=sqs.get_queue_url(QueueName
='CloudComputing')

sqs.send_message(QueueUrl=response['
QueueUrl',MessageBody='Finished!')
```

Figure 6: Send a message to the queue.

Each instance will check if there are messages in the *CloudComputing* queue. If there is any message in the queue, it proves that another instance has already found a golden nonce. Other running instances will exit proof-of-work script and move to the next steps [see Figure 7].

```
messages = sqs.receive_message(
      QueueUrl=response['QueueUrl']
)
if 'Messages' in messages:
      break
```

Figure 7: Receive and check messages.

In order to save resources, CND system will delete the *CloudComputing* queue after all instances stop finding the golden nonce. Figure 8 displays this task.

```
sqs.delete_queue(QueueUrl=response['
QueueUrl'])
```

Figure 8: Delete the queue.

VI. Amazon S3

Amazon Simple Storage System (Amazon S3) is a service for storing infinite objects. Clients can remotely read and update S3 objects using Boto3 SDK[3].

There is an existing S3 bucket named *cloudcomputinglogs* in S3 web service. When all instances stop finding the golden nonce, the local machine will immediately download log files from each instance and store log files in a local folder named *logs*. The local machine starts to upload *logs* folder to the *cloudcomputinglogs* bucket after it downloads all log files from instances.

The `upload_logs()` function in Figure 9 shows that Boto3 connects to S3 and uploads whole *logs* folder to the existing S3 bucket with the name of *cloudcomputinglogs*.

```
def upload_logs():

  s3_resource=boto3.resource("s3",
  region_name="us-east-1")

  try:
      bucket_name="cloudcomputinglogs"
      root_path = './logs'
      my_bucket=s3_resource.Bucket(bucke
  t_name)

  for path,subdirs,files in
  os.walk(root_path):

     directory_name=path.replace(root_pa
     th,"logs")

     for file in files:
       my_bucket.upload_file(os.path.joi
       n(path,file),directory_name+'/'+f
       ile)

  except Exception as err:
          print(err)
```

Figure 9: Upload a local folder to a S3 bucket.

VII. Fabric

Fabric library is used to interact with virtual machines over SSH and this tool can also execute a large number of shell commands on remote VMs.

CND system has to use N VMs in parallel for searching the golden nonce. @parallel decorator from Fabric enables N VMs to run tasks at the same time. In CND system, different instances need to test different nonces which require the system to sort instances.

As shown in Figure 10, *ihosts* array is equal to *MyHosts* array which is mentioned in the above Amazon EC2 section. Env.host represents the public ip address of the current host. The serial number of each instance can be known through a loop and the parameter LowRange is assigned to the serial number for the current instance.

```
for num in range(int(N)):
    if env.host == ihosts[num]:
        LowRange = num
```

Figure 10: Sort instances.

Next, Fabric puts the Proof-of-work script to each instance [see Figure 11].

```
put('PoW.py', 'PoW.py', use_sudo=True)
```

Figure 11: Upload pow script to each instance

According to Figure 12, CND system tries to find the golden nonce by brute-force and each instance will be distributed different values to test. The range step size is set to N (the number of current instances) which prevents that the same value is evaluated twice or more. Then, Fabric calls `run()` function to enable VMs to do Proof-of-work task.

Lastly, one instance finds the golden nonce and sends a message to other instances via SQS. At this moment, Fabric will call `get()` function which can be seen in Figure 13 to download log files from all instances and store log files in a local folder.

```
for CurrentNum in
range(int(LowRange),2**32,int(N)):
    Getvalue= run("python3 -c 'import
PoW;PoW.goldennonce(%d,%s)'"%(int(D),str(C
urrentNum)))
```

Figure 12: fabric call run() function to enable instances to do PoW.

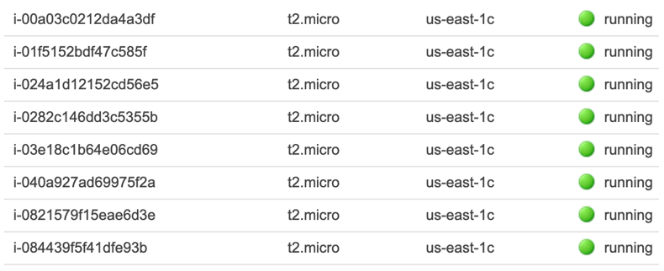
```
get('/var/log/syslog','logs/Instance%d.log
'%int(LowRange), use_sudo=True)
```

Figure 13: fabric call get() function to download log files.

VIII. The entire process of CND system

In this section, we will take T=20, D=2, N=8 as an example to show the running process of CND system.

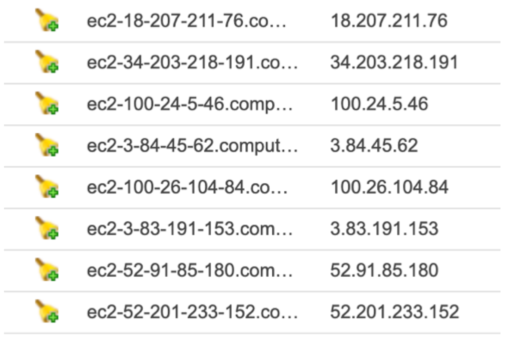
Users input the fab command: *fab -f CND.py start: T=200, D=2, N=8*, CND system will start to execute the code. Figure 14 demonstrates that CND system has already started up 8 instances.



i-00a03c0212da4a3df	t2.micro	us-east-1c	● running
i-01f5152bdf47c585f	t2.micro	us-east-1c	● running
i-024a1d12152cd56e5	t2.micro	us-east-1c	● running
i-0282c146dd3c5355b	t2.micro	us-east-1c	● running
i-03e18c1b64e06cd69	t2.micro	us-east-1c	● running
i-040a927ad69975f2a	t2.micro	us-east-1c	● running
i-0821579f15eae6d3e	t2.micro	us-east-1c	● running
i-084439f5f41dfe93b	t2.micro	us-east-1c	● running

Figure 14: CND system starts up 8 VMs.

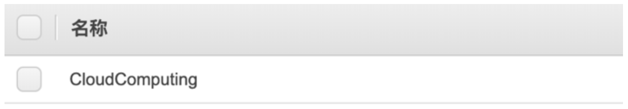
On the right side of the AWS EC2 interface that can be seen in Figure 15, it is clear that different instances have different public IP addresses.



🟨	ec2-18-207-211-76.co…	18.207.211.76
🟨	ec2-34-203-218-191.co…	34.203.218.191
🟨	ec2-100-24-5-46.comp…	100.24.5.46
🟨	ec2-3-84-45-62.comput…	3.84.45.62
🟨	ec2-100-26-104-84.co…	100.26.104.84
🟨	ec2-3-83-191-153.com…	3.83.191.153
🟨	ec2-52-91-85-180.com…	52.91.85.180
🟨	ec2-52-201-233-152.co…	52.201.233.152

Figure 15: Different instances have different public IP addresses.

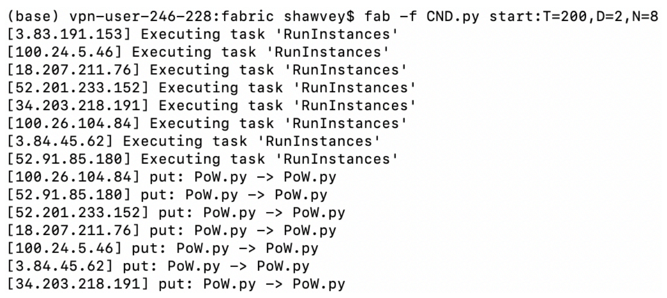
After creating the instances, the system will create an SQS queue named *CloudComputing* (see Figure 16).



名称
CloudComputing

Figure 16: CND system creates the SQS queue.

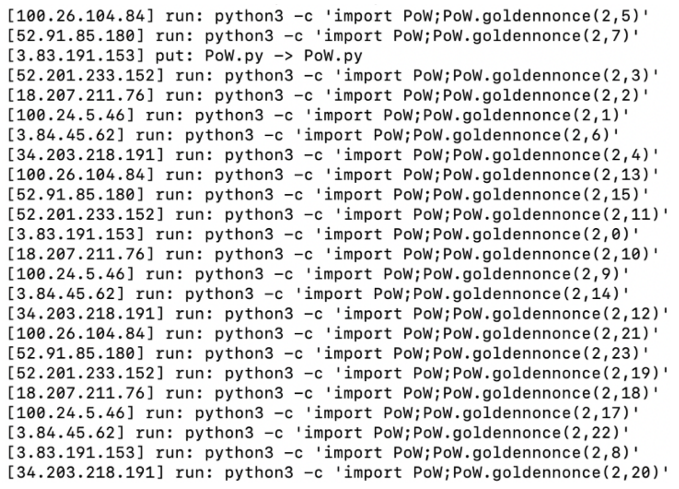
The CND system will output a large number of commands that instances are executing. In Figure 17, all instances start to execute the task in parallel and CND system uploads Proof-of-Work script to each instance via Fabric.



```
(base) vpn-user-246-228:fabric shawvey$ fab -f CND.py start:T=200,D=2,N=8
[3.83.191.153] Executing task 'RunInstances'
[100.24.5.46] Executing task 'RunInstances'
[18.207.211.76] Executing task 'RunInstances'
[52.201.233.152] Executing task 'RunInstances'
[34.203.218.191] Executing task 'RunInstances'
[100.26.104.84] Executing task 'RunInstances'
[3.84.45.62] Executing task 'RunInstances'
[52.91.85.180] Executing task 'RunInstances'
[100.26.104.84] put: PoW.py -> PoW.py
[52.91.85.180] put: PoW.py -> PoW.py
[52.201.233.152] put: PoW.py -> PoW.py
[18.207.211.76] put: PoW.py -> PoW.py
[100.24.5.46] put: PoW.py -> PoW.py
[3.84.45.62] put: PoW.py -> PoW.py
[34.203.218.191] put: PoW.py -> PoW.py
```

Figure 17: instances executes task in parallel.

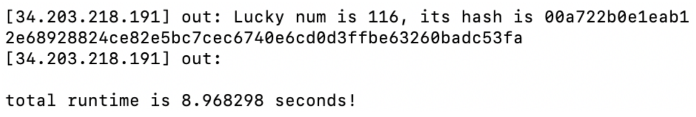
Then, instances begin to do the Proof-of-Work task. As can be seen in Figure 18, each instance tests one nonce at a time, instances can not test next nonce until current nonce is evaluated.



```
[100.26.104.84] run: python3 -c 'import PoW;PoW.goldennonce(2,5)'
[52.91.85.180] run: python3 -c 'import PoW;PoW.goldennonce(2,7)'
[3.83.191.153] put: PoW.py -> PoW.py
[52.201.233.152] run: python3 -c 'import PoW;PoW.goldennonce(2,3)'
[18.207.211.76] run: python3 -c 'import PoW;PoW.goldennonce(2,2)'
[100.24.5.46] run: python3 -c 'import PoW;PoW.goldennonce(2,1)'
[3.84.45.62] run: python3 -c 'import PoW;PoW.goldennonce(2,6)'
[34.203.218.191] run: python3 -c 'import PoW;PoW.goldennonce(2,4)'
[100.26.104.84] run: python3 -c 'import PoW;PoW.goldennonce(2,13)'
[52.91.85.180] run: python3 -c 'import PoW;PoW.goldennonce(2,15)'
[52.201.233.152] run: python3 -c 'import PoW;PoW.goldennonce(2,11)'
[3.83.191.153] run: python3 -c 'import PoW;PoW.goldennonce(2,0)'
[18.207.211.76] run: python3 -c 'import PoW;PoW.goldennonce(2,10)'
[100.24.5.46] run: python3 -c 'import PoW;PoW.goldennonce(2,9)'
[3.84.45.62] run: python3 -c 'import PoW;PoW.goldennonce(2,14)'
[34.203.218.191] run: python3 -c 'import PoW;PoW.goldennonce(2,12)'
[100.26.104.84] run: python3 -c 'import PoW;PoW.goldennonce(2,21)'
[52.91.85.180] run: python3 -c 'import PoW;PoW.goldennonce(2,23)'
[52.201.233.152] run: python3 -c 'import PoW;PoW.goldennonce(2,19)'
[18.207.211.76] run: python3 -c 'import PoW;PoW.goldennonce(2,18)'
[100.24.5.46] run: python3 -c 'import PoW;PoW.goldennonce(2,17)'
[3.84.45.62] run: python3 -c 'import PoW;PoW.goldennonce(2,22)'
[3.83.191.153] run: python3 -c 'import PoW;PoW.goldennonce(2,8)'
[34.203.218.191] run: python3 -c 'import PoW;PoW.goldennonce(2,20)'
```

Figure 18: instances do PoW task in parallel.

When one instance finds the golden nonce, it prints the golden nonce, its hash value and the total runtime [see Figure 19].

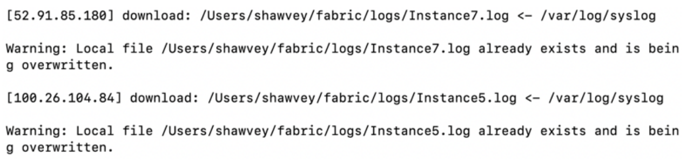


```
[34.203.218.191] out: Lucky num is 116, its hash is 00a722b0e1eab1
2e68928824ce82e5bc7cec6740e6cd0d3ffbe63260badc53fa
[34.203.218.191] out:

total runtime is 8.968298 seconds!
```

Figure 19: CND system prints the details of the golden nonce.

According to Figure 20 and Figure 21, CND system will download log files from all instances and upload those files to the S3 bucket.



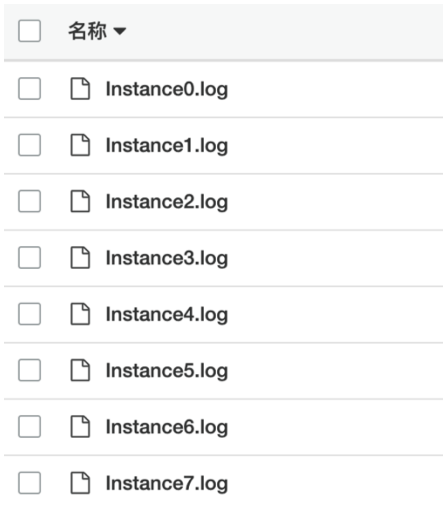
```
[52.91.85.180] download: /Users/shawvey/fabric/logs/Instance7.log <- /var/log/syslog

Warning: Local file /Users/shawvey/fabric/logs/Instance7.log already exists and is bein
g overwritten.

[100.26.104.84] download: /Users/shawvey/fabric/logs/Instance5.log <- /var/log/syslog

Warning: Local file /Users/shawvey/fabric/logs/Instance5.log already exists and is bein
g overwritten.
```

Figure 20: CND system will download log files from instances.


名称

Instance0.log
Instance1.log
Instance2.log
Instance3.log
Instance4.log
Instance5.log
Instance6.log
Instance7.log

Figure 21: S3 bucket named cloudcomputinglogs has been uploaded eight log files.

Finally, CND system will shut down those created EC2 instances and SQS queue which means our CND system is successfully executed.

IX. Conclusion

The current CND system can successfully implement a large number of useful functions, such as controlling instances, running commands remotely in parallel, getting log files from instances and so on.

However, there are still some drawbacks on this system. First of all, AWS just allows users to create eight or fewer instances by using Boto3. AWS will shut down some instances when trying to create more than 8 instances. Secondly, the speed of searching the golden nonce is slow because CND system needs to check if the SQS queue received messages and if there was a timeout while one instance receives a nonce. Thirdly, CND system spends time sending a message to the SQS queue. At the same time, other instances are still searching the golden nonce which causes that other instances cannot instantly stop searching when one instance finds the golden nonce.

There are two future research directions for this CND system. On the one hand, the author expects to build a function which recognizes the status of instances. This function will keep CND system from waiting fixed time. On the other hand, CND system can not now automatically choose the number of instances for users. The Auto Scaling Groups service is supposed to achieve this function in the future.

References

[1] Anon., (2019). Boto 3 Documentation [online]. Anon. [Viewed 20 November 2019]. Available from: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

[2] Diginmotion., (2010). Running a website on Amazon EC2 [online]. Diginmotion. [Viewed 1 December 2019]. Available from: https://s3.amazonaws.com/academia.edu.documents/27741916/running-a-website-on-amazon-ec2.pdf

[3] Matthias Brantner, Daniela Florescu, David Graf, Donald Kossmann, Tim Kraska, Building a database on S3, Proceedings of the 2008 ACM SIGMOD international conference on Management of data, June 09-12, 2008, Vancouver, Canada [doi>10.1145/1376616.1376645]