Distributed password cracking of short md5 hashes

Authors: Chun (Desmond) To Pun, Brian Sohn, Zongxi (Alex) Cheng, Megan Chen
Github link: https://github.com/meganchen/geni-mini
GENI slice name: geni-mini-cbam

Problem + Objectives

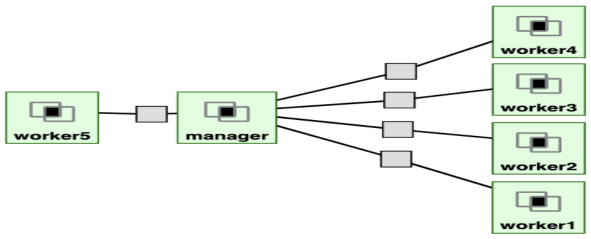
In this project, we want to understand whether it's possible to speed up the brute force search via creating a network of worker nodes. Given an md5 hash of a 5-character password (a-z, A-Z), we consider the average time it takes a hacker to recover a password. The hacker may run a distributed system in which worker nodes are dynamically activated depending on worker availability.

We design and implement a distributed system that allows dynamic participation of workers. We also collect experimental data on the average runtime for when the adversary's distributed system runs in GENI. We describe these in detail in the coming sections.

Experimental methodology

System overview. A user inputs a md5 hash of a 5-character password to the front end (in the Front node). Then, the management server enlists the help of worker nodes to help brute-force crack the password.

Network topology. We have 5 client nodes (4 workers, 1 front end) and 1 node for management service (server) on GENI with the following topology:



The front end, which receives hashes to crack, is on worker5. The network manager is on the manager node. The worker nodes are on workers 1-4 and run the brute-force algorithm for cracking the password.

Component descriptions. We give a high-level description of the user, management, and worker code.
Front end - FrontEnd.py (client):

1. Sends a TCP request to the management node (server) to establish a connection. As part of the request, identify itself as the user client, in order to distinguish between user and worker clients.
2. A user inputs a hash of a 5-character password.
3. Send the md5 hashes to the management service.

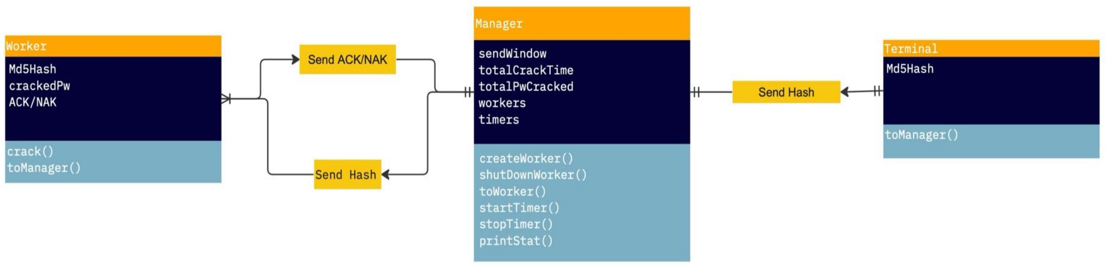
Management service - management.py (server)

1. Receive TCP requests from all worker nodes (clients) to establish a connection.
2. Receive the number of workers/password and the md5 hash from the user client.
3. Picks worker nodes to activate (see "Manager worker allocation algorithm." Section below for full description).
4. Sends the md5 hash to activated worker nodes.
5. Receives either a cracked passwords or SEARCH ENDED from worker nodes
6. If the password is cracked, print the cracking time.

Worker - worker.py (client)

1. Sends a TCP request to the management node (server) to establish a connection.
2. Runs the brute-force algorithm for cracking the md5 hash.
3. Sends the cracked password via TCP connection back to the management service.

The UML diagram is shown below.



Manager worker allocation algorithm(with dynamic allocation implemented).
The manager first asks the front end how the user wants to split each task. The manager then splits each task into parts according to user input, and waits for workers to pick up the job. Whenever a worker is available, the manager sends it a portion of the task. A task is marked allocated when all portions of it are sent to workers, then the portions of the next task are being sent subsequently. A task is marked finished when a worker returns a found password to the manager, the other portion of the task will not be sent to another worker if they have not been sent yet. When a worker finishes its job, it will notify the manager and the manager will send it the next available job.

Experiments.

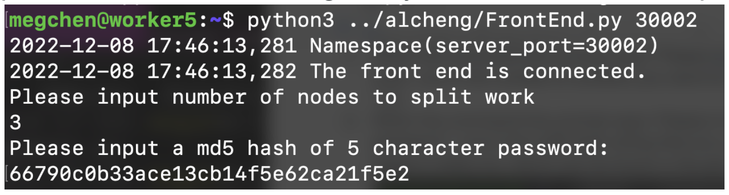
- Independent variable is the number of worker nodes sharing the brute-force workload. Dependent variable is the time to crack 1 password.

Results

Usage instructions

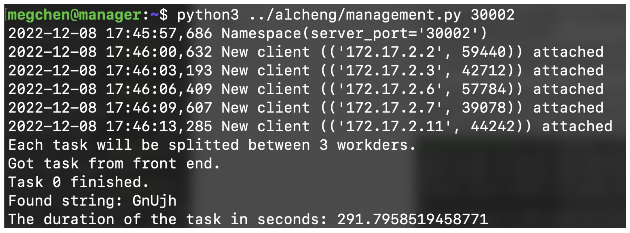
Our code runs with the following steps:

1. **Set manager port number.** Pick a <management_port_number> for the manager. This will be the same for all the commands below. For example, Megan picked <management_port_number> = 30001.
2. **Set up the management node.**
    a. ssh into the node with the command ssh -i <local ssh key file> <username>@<hostname> -p <port_number>. For example, Megan ran the command ssh -i ~/.ssh/id_geni_ssh_rsa megchen@pc2.instageni.wisc.edu -p 25010.
    b. Run management.py code: python3 ../alcheng/management.py <management_port_number>.
3. **Set up the worker nodes.** For all worker nodes (worker1 - worker 4), do the following:
    a. ssh into each worker node with the command ssh -i <local ssh key file> <username>@<hostname> -p <port_number>.
    b. Run worker.py code: python ../alcheng/worker.py <worker_port_number>.
4. **Input hash into the front end (i.e. worker5).**
    a. ssh into the worker5 node with the command ssh -i <local ssh key file> <username>@<hostname> -p <port_number>.
    b. Run FrontEnd.py code: python ../alcheng/FrontEnd.py <managment_port_number>.
    c. When the command line prompt says ‘Please input number of nodes to split work’, put in the number of workers you want to crack the password (between 1-4).
    d. When the command line prompt says ‘Please input a md5 hash of 5 character password:’, put in the md5 hash string that you want to crack. We provided test cases in tests.csv.



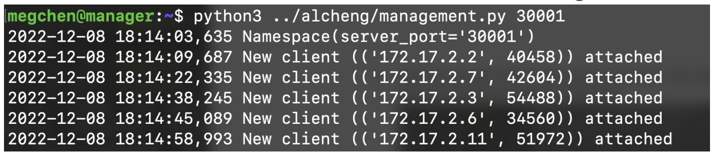
```
megchen@worker5:~$ python3 ../alcheng/FrontEnd.py 30002
2022-12-08 17:46:13,281 Namespace(server_port=30002)
2022-12-08 17:46:13,282 The front end is connected.
Please input number of nodes to split work
3
Please input a md5 hash of 5 character password:
66790c0b33ace13cb14f5e62ca21f5e2
```

    e. After the workers finish cracking the password, the manager node will print out how long it took to crack the password.



```
megchen@manager:~$ python3 ../alcheng/management.py 30002
2022-12-08 17:45:57,686 Namespace(server_port='30002')
2022-12-08 17:46:00,632 New client (('172.17.2.2', 59440)) attached
2022-12-08 17:46:03,193 New client (('172.17.2.3', 42712)) attached
2022-12-08 17:46:06,409 New client (('172.17.2.6', 57784)) attached
2022-12-08 17:46:09,607 New client (('172.17.2.7', 39078)) attached
2022-12-08 17:46:13,285 New client (('172.17.2.11', 44242)) attached
Each task will be splitted between 3 workders.
Got task from front end.
Task 0 finished.
Found string: GnUjh
The duration of the task in seconds: 291.7958519458771
```

f. After all the clients are connected, the manager command line should look like:

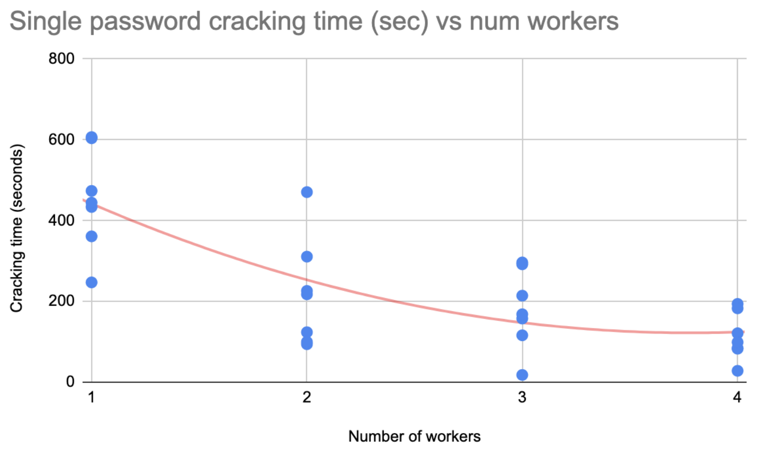


```
megchen@manager:~$ python3 ../alcheng/management.py 30001
2022-12-08 18:14:03,635 Namespace(server_port='30001')
2022-12-08 18:14:09,687 New client (('172.17.2.2', 40458)) attached
2022-12-08 18:14:22,335 New client (('172.17.2.7', 42604)) attached
2022-12-08 18:14:38,245 New client (('172.17.2.3', 54488)) attached
2022-12-08 18:14:45,089 New client (('172.17.2.6', 34560)) attached
2022-12-08 18:14:58,993 New client (('172.17.2.11', 51972)) attached
```

g. To change the number of worker nodes, kill the manager first because it is the server[1]. Next, kill the workers and front end. Finally, restart at Step 1 above.

Analysis

In our experiment, we wanted to learn if having more workers means the passwords are cracked faster. We generated the below graph on google sheets. Each blue dot represents a single password; the cracking time is shown on the Y axis and the number of workers is on the X axis. The red line is a trendline (of degree 2) for all of the data.



The cracking time decreased, on average, as the number of workers increased. This implies that the application is highly scalable as a function of the number of workers.

Our experiment works even for very limited network paths; for each password, the largest message sent in the system is a 32-character string (32 bytes). All other messages are small (an integer, a 5 character string, or 0). Further, all workers are in the same InstaGENI LAN hosted by University of Wisconsin, so our time measurements should be dominated by the time to brute-force the hash rather than network timings.

We also conducted a network analysis using the $iperf command to analyze TCP flows. When looking at the screenshot of the TCP bandwidth performance report (refer to github/exp1 and exp2), size of the transfer and bandwidth stayed mostly the same when giving back tasks to each worker and worker asking for more work to the manager. However, when a worker found the correct password, the size of the transfer increased almost tenfold. In the second experiment when we tried with multiple passwords, similar events occurred. This is due to our management algorithm (which is explained above). Our algorithm is fairly naive in that it doesn't increase the RTT or Throughput too much even when the number of tasks is increased.

[1] If we kill the front end first, this will cause the manager to incorrectly read in a bunch of inputs.

Conclusion

We learned that in the distributed system that we designed, having more workers means it takes less time to brute-force a 5 character md5 hash. Our experiment shows that short passwords are easy to break in minutes. It highlights why websites limit the password to a handful times. Also, it shows that distributing computation among several workers in parallel reduces overall computation times.

Division of labor

Architecture diagram(s) - **Alex, Chun**
Set up GENI slices and reserve resources - **Everyone**
Coding

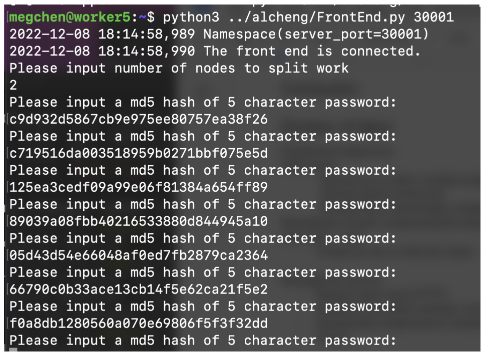
- worker.py - **Alex, Chun, Megan**
- Management.py and FrontEnd.py - **Alex, Chun**
- Create csv file of md5 test cases - **Megan, Brian**

Run experiments, Demo video, Writing report - **Megan**
Make graph - **Brian, Megan**

Appendix

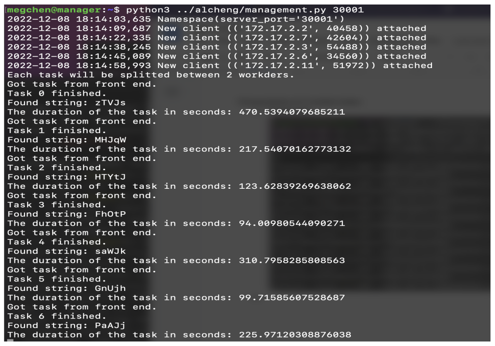
Screenshots of running the 2 worker / 1 password tests:

Front end



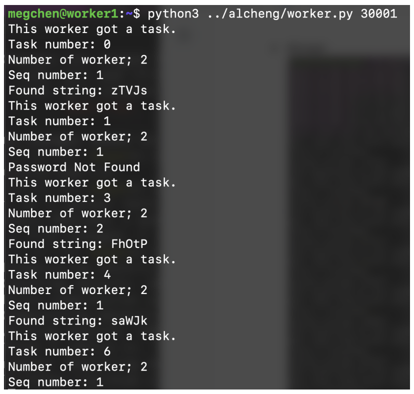
```
megchen@worker5:~$ python3 ../alcheng/FrontEnd.py 30001
2022-12-08 18:14:58,989 Namespace(server_port=30001)
2022-12-08 18:14:58,990 The front end is connected.
Please input number of nodes to split work
2
Please input a md5 hash of 5 character password:
c9d932d5867cb9e975ee80757ea38f26
Please input a md5 hash of 5 character password:
c719516da003518959b0271bbf075e5d
Please input a md5 hash of 5 character password:
125ea3cedf09a99e06f81384a654ff89
Please input a md5 hash of 5 character password:
89039a08fbb40216533880d844945a10
Please input a md5 hash of 5 character password:
05d43d54e66048af0ed7fb2879ca2364
Please input a md5 hash of 5 character password:
66790c0b33ace13cb14f5e62ca21f5e2
Please input a md5 hash of 5 character password:
f0a8db1280560a070e69806f5f3f32dd
Please input a md5 hash of 5 character password:
```

Manager



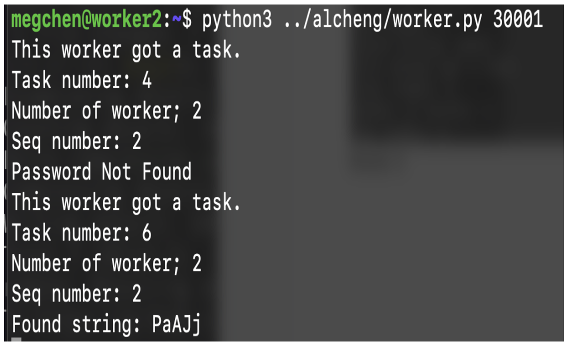
```
megchen@manager:~$ python3 ../alcheng/management.py 30001
2022-12-08 18:14:03,635 Namespace(server_port='30001')
2022-12-08 18:14:09,687 New client (('172.17.2.2', 40458)) attached
2022-12-08 18:14:22,335 New client (('172.17.2.7', 42604)) attached
2022-12-08 18:14:38,245 New client (('172.17.2.3', 54488)) attached
2022-12-08 18:14:45,089 New client (('172.17.2.6', 34560)) attached
2022-12-08 18:14:58,993 New client (('172.17.2.11', 51972)) attached
Each task will be splitted between 2 workders.
Got task from front end.
Task 0 finished.
Found string: zTVJs
The duration of the task in seconds: 470.5394079685211
Got task from front end.
Task 1 finished.
Found string: MHJqW
The duration of the task in seconds: 217.54070162773132
Got task from front end.
Task 2 finished.
Found string: HTYtJ
The duration of the task in seconds: 123.62839269638062
Got task from front end.
Task 3 finished.
Found string: FhOtP
The duration of the task in seconds: 94.00980544090271
Got task from front end.
Task 4 finished.
Found string: saWJk
The duration of the task in seconds: 310.7958285808563
Got task from front end.
Task 5 finished.
Found string: GnUjh
The duration of the task in seconds: 99.71585607528687
Got task from front end.
Task 6 finished.
Found string: PaAJj
The duration of the task in seconds: 225.97120308876038
```

Worker 1:



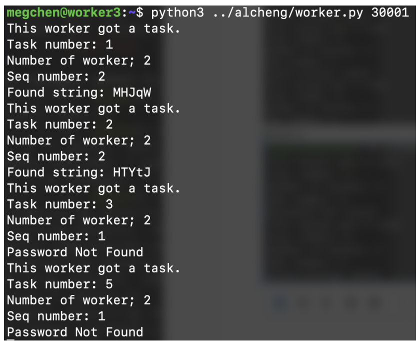
```
megchen@worker1:~$ python3 ../alcheng/worker.py 30001
This worker got a task.
Task number: 0
Number of worker; 2
Seq number: 1
Found string: zTVJs
This worker got a task.
Task number: 1
Number of worker; 2
Seq number: 1
Password Not Found
This worker got a task.
Task number: 3
Number of worker; 2
Seq number: 2
Found string: FhOtP
This worker got a task.
Task number: 4
Number of worker; 2
Seq number: 1
Found string: saWJk
This worker got a task.
Task number: 6
Number of worker; 2
Seq number: 1
```

Worker 2:



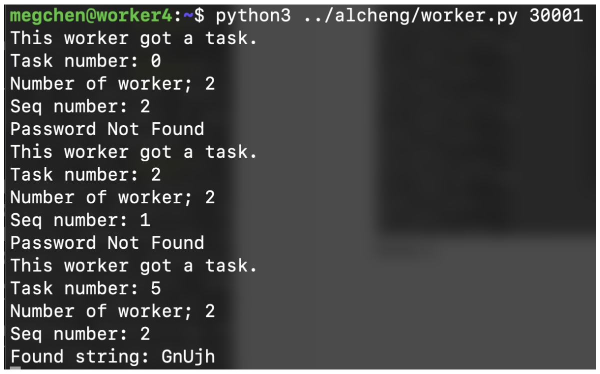
```
megchen@worker2:~$ python3 ../alcheng/worker.py 30001
This worker got a task.
Task number: 4
Number of worker; 2
Seq number: 2
Password Not Found
This worker got a task.
Task number: 6
Number of worker; 2
Seq number: 2
Found string: PaAJj
```

Worker 3:



```
megchen@worker3:~$ python3 ../alcheng/worker.py 30001
This worker got a task.
Task number: 1
Number of worker; 2
Seq number: 2
Found string: MHJqW
This worker got a task.
Task number: 2
Number of worker; 2
Seq number: 2
Found string: HTYtJ
This worker got a task.
Task number: 3
Number of worker; 2
Seq number: 1
Password Not Found
This worker got a task.
Task number: 5
Number of worker; 2
Seq number: 1
Password Not Found
```

Worker 4:



```
megchen@worker4:~$ python3 ../alcheng/worker.py 30001
This worker got a task.
Task number: 0
Number of worker; 2
Seq number: 2
Password Not Found
This worker got a task.
Task number: 2
Number of worker; 2
Seq number: 1
Password Not Found
This worker got a task.
Task number: 5
Number of worker; 2
Seq number: 2
Found string: GnUjh
```