Master-Workers Based Distributed Password Cracker

Yufeng Chen, Weixi Li, Zijian Lin, Ruihong Zhu

Github Link: https://github.com/MushroomLin/DistributedPasswordCracker
Project on Geni: Slice Name: PAssWoRdCrAcker.

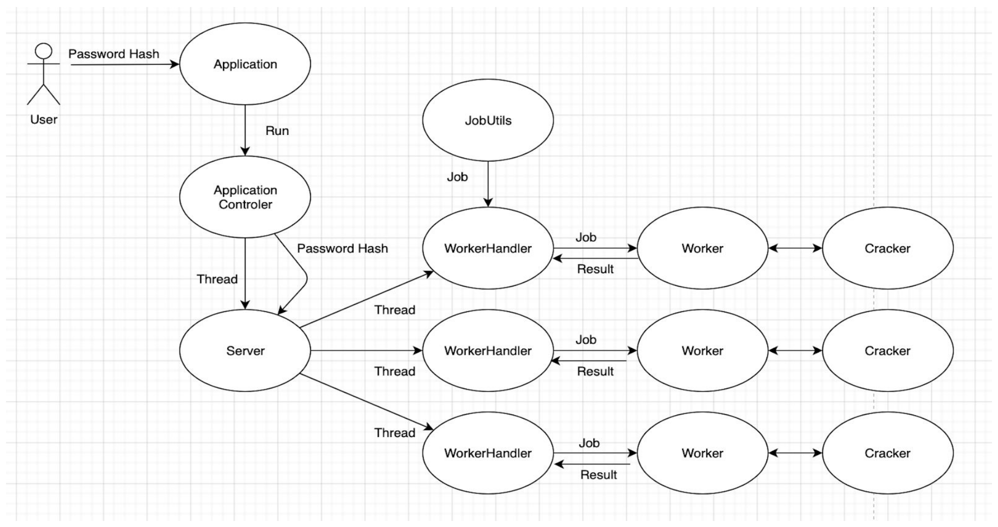
Introduction

The goal of this project is to design a distributed password cracker distributed which allows concurrent password cracking on multiple server machines. The system will be able to crack md5 hashed 5-character password (a-z, A-Z) submitted by users. Users can enter a hashed string by using our web interface and the outcome will be the correct unhashed 5-character password corresponding to this string.
The learning outcomes are the correctness of our cracker, the change of running by using multiple workers in parallel and possible ways to further enhance our application.

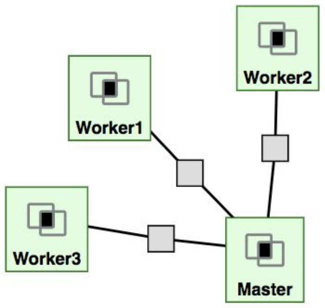
Experimental Methodology

In our system, we will have one master node which will maintain a password job queue and dispatch password cracking jobs to workers. The system will have multiple workers cracking the password concurrently and will support adding/removing workers on the fly. The workers will brute force the password locally and return the result to the master node if the correct password is found.
The working flow of our system should be: First, the Application.java file needs to be run to start the SpringBoot application. Next, Users can open a browser and request /cracking in their localhost. Finally, the Worker.java file needs to be started.
The cracking request will be received by ApplicationHandler.java and it will create a new thread as a server. The ApplicationController will continue to run and pass the hashed string to the server. The Server.java file will create a new generator and a socket to receive the hashed string. The Generator.java file has three parameters: base, maxChars, and rangeSize. In our case, the base will equal to 52 since we have to handle both capital and non-capital letters. The maxChars will equal to 5 because our password's maximum digit number is 5. The rangeSize represents a subset of the entire possible combination of our password. The number of possible combinations is too large so that we have to divide it into different ranges and crack them accordingly. Workers are defined by Worker.java file and each worker will process a certain range of possible combinations. It will output a "FOUND" message if it is able to crack the hashed string within this range, or a "REQUEST_RANGE" message if it didn't find it.

These messages will be received by Clienthandler.java and it will either finished the program or assign a new range to workers according to which message is received. The Worker.java uses Cracker.java to crack hashed strings with brute force. It will hash all possible combinations and compare it with our entered hash string. If they are the same, then we found the correct password. We used an open-source library (MD5.java, MD5State.java, MD5InputStream.java) to speed up the hashing process.



There is one master node and three worker nodes in our Geni slice. Therefore, the maximum worker number is 3.

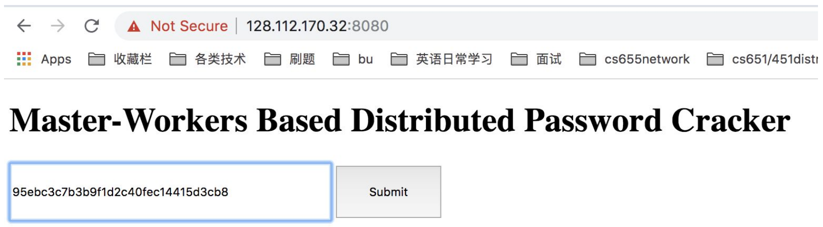


Results

Usage Instruction

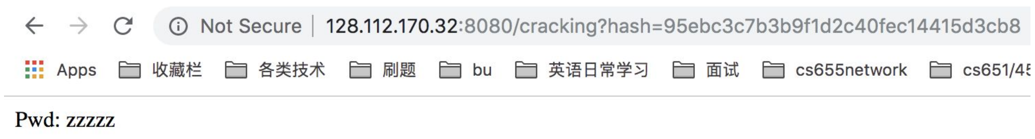
1. Enter Master node on Geni, type “java -jar password-cracker-0.1.1.jar” to start the application.
2. Open a browser, enter the IP address of the master node, in our case, it is “128.112.170.32:8080”.
3. Enter the hashed string you wish to crack in the web interface and click the submit button. We used “95ebc3c7b3b9f1d2c40fec14415d3cb8”, the hash version of zzzzz to test our application.

4. Enter Worker node, type “javac -d ./ *.java” to compile the code. Type “java passwordcracker.Worker” to start a worker. You can use this method to start multiple workers to process parallelly. You can also run our Worker.java file locally by changing its String Host to the IP address of the master node.
5. There will be a timer to record the running time in ms.
6. We conduct five experiments for each different number of the worker.



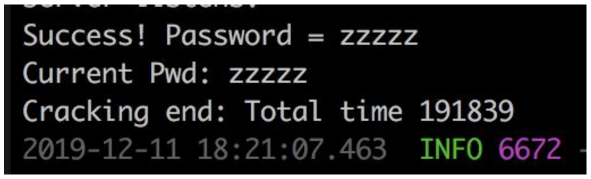
Master-Workers Based Distributed Password Cracker

95ebc3c7b3b9f1d2c40fec14415d3cb8



Pwd: zzzzz

One Worker



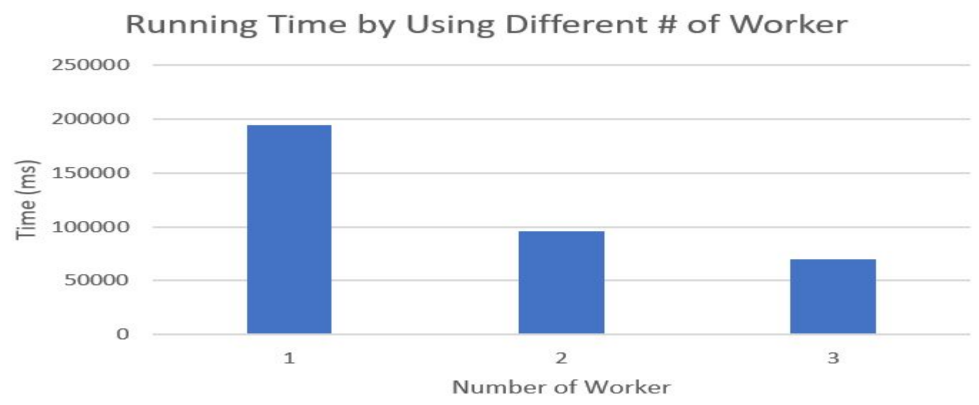
```
Success! Password = zzzzz
Current Pwd: zzzzz
Cracking end: Total time 191839
2019-12-11 18:21:07.463  INFO 6672 -
```

Analysis

Worker 1 time: {191839, 204834, 184628, 194694, 196958} Average = 194590.6

Worker 2 time: {94121, 106450, 97862, 86629, 95743} Average = 96161

Worker 3 time: {73386, 63317,74768, 65082, 72243} Average = 69759.2



Our results show that our cracker is able to get the correct password. The running time will be significantly reduced by using more than one worker. Using two workers will reduce the running time by half and using three workers will reduce the running time by ⅔. The reduce rate is almost linear, which shows that our workers are able to work parallelly and perform well.

Conclusion

Our main finding is that there are many possible methods to optimize the performance of an application. In our case, we used the MD5 open library to speed up the hashing process, and all possible combinations will be divided into different ranges and assigned to different workers. These methods significantly lower the running time of our application.

However, there are still many ways to further expand our application. For example, if the master node is accidentally down, it cannot restart itself and all the calculation process will be lost. If a working node is down, then it's assigned ranges will not be transferred to other workers. Therefore, we can further enhance our cracker's ability to handle mistakes and edge cases.

Another issue is that we need to start the web interface first before running workers, which can be modified so that workers will be started as soon as the application started. We can also modify the web interface to make it more friendly to users.

Division of Labor

Yufeng wrote the Application Controller part and designed the experiment and wrote the Results section of our report.

Weixi designed the job utlis and created Geni slice and deployed our code to Geni.

Zijian designed the code structure and wrote the codes of Worker and WorkerHandler.java.

Ruihong wrote the cracker.java, the web interface and implement the MD5 library. He also wrote the Experimental Methodology section of the report.