**Distributed Hash Cracker**

**A PROJECT REPORT**

*Submitted by*

Adithya Murali,15BCE0012

Aditya Jain,15BCE0250

Harsha Vardhan Reddy 15BCE0721

Saloni Srivastava,15BCE2033

Course Code: CSE4001

Course Title: Parallel and Distributed Computing

*In partial fulfilment for the award of the degree of*

**B.Tech**

**in**

**Computer Science and Engineering**

Under the guidance of

**Prof. Srimathi C.**

**Associate Professor, SCOPE,**

**VIT University, Vellore.**



**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**NOVEMBER, 2017**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Topic** | | **Page No.** |
| 1 | Introduction | 1.1 Hashing | 4 |
| 1.2 Hash Cracking | 4 |
| 1.3 Gearman | 5 |
| 1.4 How does Gearman work | 5 |
| 2 | Overview of the Work | 2.1 Problem Description | 6 |
| 3 | Implementation | 3.2 Source Code | 8 |
| 3.3 Test Cases | 10 |
| 3.4 Execution Snapshots | 10 |
| 4 | Conclusion and Future Directions | | 11 |
| 5 | References | | 11 |

**ABSTRACT**

The web is constantly evolving with new technologies being added all the time, creating a platform completely unrecognisable from when the web first began. Recovery of passwords protected by one-way hashes is a problem ideally suited to parallel computing, due to the embarrassingly parallel nature of a brute force attack. Although many computer forensics and penetration testing tools can perform multithreaded hash cracking on SMP systems, modern iterated-hash techniques require unacceptably long crack time on a single computer. This presents a distributed hash cracking system capable of running on all major platforms.

The main aim behind the project was to try to harness the power of retrieving passwords from hashes using a dictionary attack technique. If this proved possible, the secondary aim was to assess how cost effective it would be to retrieve hashes in this way.

The rate of password guessing depends heavily on the cryptographic function used by the system to generate password hashes. Since it would take huge amount of time to crack a password on a single computer we are trying to implement it on several computers which in turn decreases the time of cracking and makes it efficient.

1. **INTRODUCTION**

**1.1 HASHING**

Hashing is the transformation of a string of [character](http://searchcio-midmarket.techtarget.com/definition/character)s into a usually shorter fixed-length value or key that represents the original string. Hashing is used to index and retrieve items in a [database](http://searchsqlserver.techtarget.com/definition/database) because it is faster to find the item using the shorter hashed key than to find it using the original value. It is also used in many [encryption](http://searchsecurity.techtarget.com/definition/encryption) algorithms.

The hashing algorithm is called the *hash function--* probably the term is derived from the idea that the resulting hash value can be thought of as a "mixed up" version of the represented value.

The hash function is used to index the original value or key and then used later each time the data associated with the value or key is to be retrieved. Thus, hashing is always a one-way operation. There's no need to "reverse engineer" the hash function by analyzing the hashed values. In fact, the ideal hash function can't be derived by such analysis. A good hash function also should not produce the same hash value from two different inputs. If it does, this is known as a *collision*. A hash function that offers an extremely low risk of collision may be considered acceptable.

**1.2 HASH CRACKING**

Hash cracking is the process of recovering [passwords](https://en.wikipedia.org/wiki/Password) from [data](https://en.wikipedia.org/wiki/Data_(computing)) that have been stored in or transmitted by a [computer system](https://en.wikipedia.org/wiki/Computer_system). A common approach ([brute-force attack](https://en.wikipedia.org/wiki/Brute-force_attack)) is to try guesses repeatedly for the password and check them against an available [cryptographic hash](https://en.wikipedia.org/wiki/Cryptographic_hash_function) of the password.

The purpose of password cracking might be to help a user recover a forgotten password (installing an entirely new password is less of a security risk, but it involves System Administration privileges), to gain unauthorized access to a system, or as a preventive measure by [system administrators](https://en.wikipedia.org/wiki/System_administrator) to check for easily crack-able passwords. On a file-by-file basis, password cracking is utilized to gain access to digital evidence for which a judge has allowed access but the particular file's access is restricted.

The time to crack a password is related to bit strength , which is a measure of the password's entropy, and the details of how the password is stored. Most methods of password cracking require the computer to produce many candidate passwords, each of which is checked. The rate of password guessing depends heavily on the cryptographic function used by the system to generate password hashes. Since it would take huge amount of time to crack a password on a single computer we are trying to implement it on several computers which in turn decreases the time of cracking and makes it efficient.

**1.3 GEARMAN**

Gearman is an open source application framework designed to distribute appropriate computer tasks to multiple computers, so large tasks can be done more quickly. In some cases, load balancing rather than raw speed may be the main goal; a Web server, for instance, could use Gearman to send tasks for which it is not optimized to another computer (which may be running on a different architecture, using another operating system, or loaded with a computer language better suited to a particular operation).

It has been originally written in Perl by Brad Fitzpatrick. Brian Aker and Eric Day rewrote the framework in C.

**1.4 HOW DOES GEARMAN WORK?**

Gearman assigns each involved computer a role as client, job server, or worker. A worker machine can be assigned multiple instances of the worker role, which allows more powerful computers to complete more portions of a given task. Tasks originate on a client, are transmitted from the client to the job server, and performed on one or more workers. The completed task's output is then returned, again by way of the job server, to the client where the task originated. Gearman is conceptually related to Map Reduce; Gearman handles MapReduce by allowing worker nodes to map out work to other workers, with the original worker acting as the reducer.

Gearman performs coalescence on the work sent by a client. If two or more clients ask for work to be completed on the same body of work, either by seeing that the same blocks are being sent or by using the unique value sent by the client, it will coalesce the work so that only one worker is used. It does this specifically to avoid thundering herd problems which are common to cache hit failures.

To mitigate the damage that would be done if a job server (or its network connection) were to fail, clients can be configured with more than one assigned job server; if the first assigned job server fails, another can be transparently substituted.

Gearman implements a protocol that consists of binary packets containing requests and responses; this protocol defines the structure of messages passing between the three parts of a Gearman implementation. By default, the Gearman protocol uses TCP port 4730. It previously operated on port 7003, but this conflicted with the AFS port range and the new port (4730) was assigned by IANA.

The name "Gearman" was chosen as an anagram for "Manager", "since it dispatches jobs to be done, but does not do anything useful itself."

1. **OVERVIEW OF THE WORK**

**2.1.PROBLEM DESCRIPTION**

Password cracking is the process of recovering [passwords](https://en.wikipedia.org/wiki/Password) from [data](https://en.wikipedia.org/wiki/Data_(computing)) that have been stored in or transmitted by a [computer system](https://en.wikipedia.org/wiki/Computer_system). A common approach ([brute-force attack](https://en.wikipedia.org/wiki/Brute-force_attack)) is to try guesses repeatedly for the password and check them against an available [cryptographic hash](https://en.wikipedia.org/wiki/Cryptographic_hash_function) of the password. The purpose of password cracking might be to help a user recover a forgotten password, to gain unauthorized access to a system, or as a preventive measure by [system administrators](https://en.wikipedia.org/wiki/System_administrator) to check for easily crackable passwords. The time to crack a password is related to bit strength, which is a measure of the password's [entropy](https://en.wikipedia.org/wiki/Entropy_(information_theory)), and the details of how the password is stored. Most methods of password cracking require the computer to produce many candidate passwords, each of which is checked. Since it would be very inefficient on a single computer, it could be made cost effective by distributing on multiple computers . More computers could parallelly keep checking for string of letters and it would provide effective results.

1. **IMPLEMENTATION**

**3.1.DESCRIPTON OF MODULES/PROGRAMS**

**3.2.SOURCE CODE**

**Worker**

#!/usr/bin/python

import gearman,json,hashlib

gm\_worker = gearman.GearmanWorker(['localhost:4730'])

def crack(gearman\_worker, gearman\_job):

var=gearman\_job.data

var=json.loads(var)

var['password']=var['password'].rstrip()

var['password']=var['password'].lower()

hash\_object = hashlib.sha256(var['password'])

hex\_dig = hash\_object.hexdigest()

print hex\_dig

if hex\_dig==var['hash']:

return str(var['password'])

return ""

# gm\_worker.set\_client\_id is optional

gm\_worker.set\_client\_id('python-worker1')

gm\_worker.register\_task('crack', crack)

# Enter our work loop and call gm\_worker.after\_poll() after each time we timeout/see socket activity

gm\_worker.work()

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**Client**

#!/usr/bin/python

import gearman,argparse,hashlib,json,subprocess

def check\_request\_status(job\_request):

if job\_request.complete:

print "Job %s finished! Result: %s - %s" % (job\_request.job.unique, job\_request.state, job\_request.result)

elif job\_request.timed\_out:

print "Job %s timed out!" % job\_request.unique

elif job\_request.state == JOB\_UNKNOWN:

print "Job %s connection failed!" % job\_request.uniqueparser=argparse.ArgumentParser(description='Hello')

parser.add\_argument('-w','--wordlist',help='enter path to worldlist',action='store',default=False,dest='wl')

parser.add\_argument('-f','--file',help='enter path to hash',action='store',default=False,dest='hash')

args=parser.parse\_args()

if args.wl==False:

print 'Enter wordlist (-w parameter)'

exit

if args.hash==False:

print 'Enter hash file (-f parameter)'

exit

gm\_client = gearman.GearmanClient(['localhost:4730'])

queue=list()

with open(args.hash,'r') as f:

password=f.read()

password=password.lower()

with open(args.wl,'r') as f:

for x in f:

p1={"hash":password,"password":x}

p1=json.dumps(p1)

job\_request = gm\_client.submit\_job("crack", p1)

queue.append(job\_request)

for i in queue:

check\_request\_status(i)

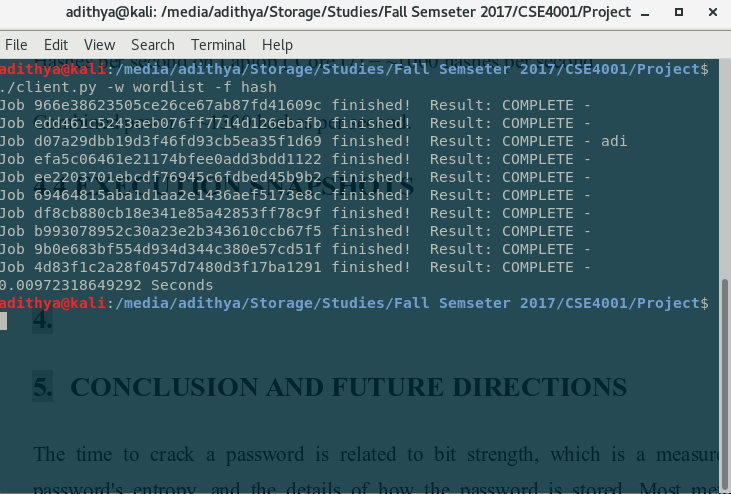
**4.3 TEST CASES**

Hashes per second on Raspberry pi = ~333 hases per second.

Hashes per second on Laptop ( Core i7) = ~1000 hashes per second.

Combined power = ~1300 hashes per second.

**4.4 EXECUTION SNAPSHOTS**

****

1. **CONCLUSION AND FUTURE DIRECTIONS**

The time to crack a password is related to bit strength, which is a measure of the password's entropy, and the details of how the password is stored. Most methods of password cracking require the computer to produce many candidate passwords, each of which is checked. Since it would be very inefficient on a single computer. Running it parallelly on several computers makes it more efficient as more computers would keep checking through256 string of letters and the word could be retrieved with better effectiveness.

1. **REFERENCES**
2. MwrInfo Security
3. Andrew Zoenberg, "Distributed Hash Cracker", May 2009
4. TechTarget