Network and Grid Computing

Distributed MD5 Hash Cracking

This assignment covers the creation of a program that is designed to pass data between machines in order to break an MD5 hash, to discover the cleartext.

2011

David Sivocha

CE00854-5 Network and Grid Computing

15/4/2011

Contents

[Introduction 3](#_Toc290627046)

[Design 4](#_Toc290627047)

[Initial Data flow between Client and Server 4](#_Toc290627048)

[Pseudo Code 4](#_Toc290627049)

[Server 4](#_Toc290627050)

[Client 5](#_Toc290627051)

[Specific Considerations 6](#_Toc290627052)

[Implementation 7](#_Toc290627053)

[Server Code 7](#_Toc290627054)

[Client Code 11](#_Toc290627055)

[Screenshots 16](#_Toc290627056)

[Server 16](#_Toc290627057)

[Client 17](#_Toc290627058)

[Discovered Issues 18](#_Toc290627059)

[Recovered Clear Text 19](#_Toc290627060)

[Conclusions 19](#_Toc290627061)

# Introduction

This assignment aims to introduce concepts built around network and grid computing in order to make use of processing power available to the users.

The aim of the assignment is to produce a server/client solution that could be used in order to brute force MD5 hashes of numbers from a value of -2,147,483,647 to 2,147,483,647.

The program itself would aim to use the network to pass out instructions to various clients, so that the actual process of checking the computed hashes versus the one given would be split over multiple machines, thus reducing the amount of time it takes to crack a hash.

The program also had to be able to have some amount of error checking, and the ability to restart from where it left off from the previous session if the server crashed.

The application will attempt to prove that distributed computing is a viable option especially when it comes to long processes that only require processing power to achieve.

The program will itself be an imitation of similar solutions such as the SETI@home project that uses distributed computing power to crunch through the data it receives by using spare processing cycles on a client’s home machine.

# Design

Initially the client program is fairly simple. It needs to have functions that can look at the hash and then check it against a pre-computed value, and do so again and again until the correct answer has been found.

The server requires more forethought, as it needs to have several sections revolving around passing packets out to the clients on request, as well as some method of tracking how far the process is along.

The client also needs instructors in it, to allow it to find the server and then request new chunks.

## Initial Data flow between Client and Server

1. Server Starts
2. Server receives hash
3. Creates logfile
4. Opens ports
5. Client pings server for response
6. Response given to client
7. Client requests chunk
8. Server provides chunk, notes in log file
9. If client gets result send yes back, if not it sends a no back
10. 2 stages
    1. If yes received, send kill signal to all clients
    2. If no received, send client new chunk
11. Repeat steps 7 – 10 until a result is returned from server

## Pseudo Code

### Server

Server {  
main{

Console.Output(Welcome Screen)  
 Console.input(hash)  
 if(logfile exists)  
 check for last number  
 if number available chunk will start from their  
 else  
 create new logfile  
   
 Listen for packets  
 if hello received  
 reply with hash  
 if no received  
 send client new chunk  
 if yes received  
 send terminate to all clients  
}  
ipaddresss{  
get a list of addresses the server can use  
}

### Client

Client{

Console.Output(Welcome Screen)  
 Console.input(Server IP Address)  
 hash = Server Connect()  
 Start Thread = ListenforTerminate  
 Ask server for chunk  
 result = md5Crack(hash, chunk)  
while ListenforTerminate.isAlive  
 if result yes  
 Tell server  
 Kill program  
 if result no  
 Request new chunk

}

ServerConnect(){

Send hello to server  
 wait for reply  
 If reply  
 Pass hash back to main  
 if no reply  
 Terminate

}

ListenforTerminate(){

Joins Multicast group  
 Listen for global terminate  
 Terminate Received = Kill Program

}

md5Crack(string hash, int chunk){

use chunk number to compute hash  
 check hash against compute hash  
 if match  
 pass yes  
 if !match  
 Pass no  
}

## Specific Considerations

The structure of the program is fairly simple. The complications come from open sockets and ports, and ensuring that the clients and the server are all talking to each other on the same ports.

To this end we need to ensure that a dedicated number of ports are set to ensure that we can refer to them during programming.

|  |  |
| --- | --- |
| Port Number | Purpose |
| 8009 | Communications sent to the server – receiving port |
| 8010 | Communications sent to the client – receiving port |
| 8012 | Multicast receiving port |

We also need to decide on a size of chunk for each machine to process before asking for another one. As the purpose of this program is to speed up the cracking of hashes, ideally smaller chunks would be more applicable, so they can be passed out to client machines quicker.

I settled on using chunks of 100000. This should be processed fairly quickly by client machines, while ensuring that the chunks aren’t so large that they take time to process.

To pass data between machines I have settled on using UDP unicasts, as they have far less overhead that TCP and overall will prove to be faster than a Bi-Directional TCP stream. While TCP will ensure that packets are less likely to be dropped, it’s a simple matter for a client machine to re-request a chunk to process.

We also need to have some way of sending a kill signal to all clients when the correct hash has been found. To this end we ideally want to use a UDP multicast, as this can be used to ensure that all the clients receive the message, as well as it being an already established method of communication as the rest of the program uses UDP unicasts already.

# Implementation

## Server Code

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Net;

using System.Net.Sockets;

using System.Threading;

using System.Net.NetworkInformation;

namespace MD5CrackerServer

{

class Server

{

static void Main(string[] args)

{

//Stores the time the server started

DateTime startTime = DateTime.Now;

//Variable to store the initial hash

String hash = "";

//Greetings Screen

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("\* Distributed MD5 \*");

Console.WriteLine("\* Cracker \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("\* Server \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("\* 09000451 \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("");

Console.WriteLine("--------------------");

Console.WriteLine("Server Has started at {0}", startTime);

Console.WriteLine();

Console.WriteLine("Please use any of the following addresses to connect to the server");

//gets the machines IP Addresses and prints them for reference

IPAddress[] Adresses = GetAllUnicastAddresses();

foreach (IPAddress Adres in Adresses){

Console.WriteLine("IP Address: {0}", Adres);

}

Console.WriteLine();

//Request Hash Input, checks to make sure field is not null

Console.WriteLine("Please Input the hash desired to be cracked");

hash = Console.ReadLine();

while (hash == "" || hash == null)

{

Console.WriteLine("No Input detected, please type in a hash!");

hash = Console.ReadLine();

}

//Section Starts the Server and waits for a return hash value

string hashValue = serverStart(hash);

Thread.Sleep(50);

//hashvalue passed onto terminate clients to be passed out to all machines

terminateClients(hashValue);

//delays the program end

Console.ReadLine();

}

//Function to get IP addresses

public static IPAddress[] GetAllUnicastAddresses(){

// By passing an empty string to GetHostEntry we receive all the IP addresses on the local machine

IPHostEntry LocalEntry = Dns.GetHostEntry("");

return LocalEntry.AddressList;

}

//this function when called sends a multicast out to all clients available telling them the hash has been found and killing their own processes

static void terminateClients(String hashValue) {

UdpClient udpClient = new UdpClient();

Byte[] sendBytes = new Byte[1024]; // buffer to read the data into 1 kilobyte at a time

Console.WriteLine("Server is Started");

IPAddress address = IPAddress.Parse("225.0.0.1"); //get mulitcast address

udpClient.Connect(address, 8012); //open a connection to that location on port 8012

string data = hashValue;

sendBytes = Encoding.ASCII.GetBytes(data.PadRight(1024));

udpClient.Send(sendBytes, sendBytes.GetLength(0)); //send information to the port

Console.WriteLine("Terminate Information Sent"); //user feedback

Console.WriteLine("Program can now terminate");

}

static string serverStart(string hash)

{

string returnData = "";

UdpClient udpClient = new UdpClient(); //udp client for sending data

UdpClient udpClient2 = new UdpClient(8009); //udp client fixed on port 8009 for receiving data

Byte[] recieveBytes = new Byte[1024]; // buffer to read the data into 1 kilobyte at a time

Byte[] sendBytes = new Byte[1024]; // buffer to read the data into 1 kilobyte at a time

IPEndPoint remoteIPEndPoint = new IPEndPoint(IPAddress.Any, 8009); //open port 8009 on this machine

String reply = null;

String splitYN = null;

String hashValue = null;

int count = 0;

DateTime startTime = DateTime.Now;

//Path to Logfile

String fileName = "c:\\09000451-log.txt";

StreamWriter Swriter; //stream to write to a logfile

StreamReader Sreader; //stream to read from logfile

if (File.Exists(fileName)) //check to see if the file exists

{

Console.WriteLine("Log File found! Continuing from last entry!");

String inputtext = null;

Sreader = new StreamReader(fileName);

while (Sreader.Peek() >= 0)

{

inputtext = Sreader.ReadLine();

}

String tempcount = inputtext.Split()[1];

try

{

//checks to see if there is at least one numerical entry in the logfile, if not, count will start at 0, but this does stop the program crashing if there is no logfile contents

count = Convert.ToInt32(tempcount);

Console.WriteLine("Starting from {0}", count);

}

catch (FormatException e)

{

Console.WriteLine("Unable to find any entries in logfile! Starting from 0!");

count = 0;

}

Sreader.Close();

}

else

{

//file does not already exist start from the begining

Console.WriteLine("No Log File found! Creating now!");

Swriter = new StreamWriter(fileName, false); //name of file

startTime = DateTime.Now;

Swriter.WriteLine("Hash {0} started on {1}", hash, startTime);

Swriter.Close();

}//end of the IF else

Swriter = new StreamWriter(fileName, true); //name of file

Console.WriteLine("Server is Started");

Console.WriteLine("");

Console.WriteLine("Clients can now connect");

//keep recieving packets until a terminate is sent from a client, in this case terminate is in the form of a yes packet sent by a machine discovering the correct hash

while (splitYN != "y")

{

recieveBytes = udpClient2.Receive(ref remoteIPEndPoint);

returnData = Encoding.ASCII.GetString(recieveBytes);

splitYN = returnData.Split()[0];

hashValue = returnData.Split()[1];

if (splitYN == "Hello")

{

//if a hello packet is detected, sends a confirm hello back with the hash attached to the datagram, so the clients recieve hashes straight away

Console.WriteLine(remoteIPEndPoint.Address.ToString() + " connected!");

IPAddress remoteAddr = remoteIPEndPoint.Address; //IP address of the server entered

udpClient.Connect(remoteAddr.ToString(), 8010); //address of the remotelocation

reply = "Hello " +hash;

sendBytes = Encoding.ASCII.GetBytes(reply.PadRight(1024));

udpClient.Send(sendBytes, sendBytes.GetLength(0)); //send the packet

}

else if (splitYN == "n")

{

//writes to the logfile to let it know what chunks have been sent

Console.WriteLine("Sending chunk to " + remoteIPEndPoint.Address.ToString());

String ip = remoteIPEndPoint.Address.ToString();

Swriter.WriteLine("Sent {0} - {1} to {2}", count, count+100000, ip);

Swriter.Flush(); // included to ensure data is written to the logfile immediately

IPAddress remoteAddr = remoteIPEndPoint.Address; //IP address of the server entered

udpClient.Connect(remoteAddr.ToString(), 8010); //address of the remotelocation

reply = count.ToString();

sendBytes = Encoding.ASCII.GetBytes(reply.PadRight(1024));

udpClient.Send(sendBytes, sendBytes.GetLength(0)); //send the packet

count = count + 100000; //increments the count ready to send the next chunk on

}

else if (splitYN == "y")

{

//once a tes packet is received these statements kick in, writing to both screen and file

DateTime endTime = DateTime.Now;

Console.WriteLine("HASH FOUND!");

Console.WriteLine("{0} converts to {1}", hash, hashValue);

Swriter.WriteLine("");

Swriter.WriteLine("Cleartext Found: the hash converst to {0}!", hashValue);

Swriter.WriteLine("");

Swriter.WriteLine("Cracking ended at {0}", endTime);

Swriter.Close();

}

}

//the actual value of the hash will not be returned until the while loop closes on receipt of a yes packet

return hashValue;

}

}

}

## Client Code

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Net;

using System.Net.Sockets;

using System.Security.Cryptography;

using System.Threading;

namespace MD5CrackerClient

{

class Client

{

static void Main(string[] args)

{

//tells the system to only use available processor cycles allowing the program to run in the background

System.Diagnostics.Process.GetCurrentProcess().PriorityClass = System.Diagnostics.ProcessPriorityClass.Idle;

//Greetings Screen

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("\* Distributed MD5 \*");

Console.WriteLine("\* Cracker \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("\* Client \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("\* 09000451 \*");

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine("");

Console.WriteLine("--------------------");

//connects to server

Console.WriteLine("What is the IP of your server");

String ServerName = Console.ReadLine();

string hash = serverConnect(ServerName);

Console.WriteLine("Received Hash! " + hash);

Thread.Sleep(50);

//starts a thread listening for multicast terminate instructions, this allows the program to continue functioning while keeping a constant listen for global instructions.

Thread terminatorThread = new Thread(new ThreadStart(terminateThread));

terminatorThread.Start();

//creates udp clients for listening!

UdpClient udpClient = new UdpClient(); //outgoing Udp

UdpClient udpClient2 = new UdpClient(8010); //incoming port

//section executes code while the thread is alive, this will include requesting new chunks to work through

String resultYN = null;

while (terminatorThread.IsAlive)

{

Byte[] sendBytes = new Byte[1024]; // buffer to read the data into 1 kilobyte at a time

Byte[] recieveBytes = new Byte[1024]; // buffer to read the data into 1 kilobyte at a time

String textinput = null;

String returnData = "";

//sends an initial No to the server to request a chunk, as the server is keyed to pass out new chunks to clients that don't have an answer.

try

{

IPAddress remoteAddr = Dns.GetHostEntry(ServerName).AddressList[0]; //IP address of the server entered

udpClient.Connect(remoteAddr.ToString(), 8009); //address of the remotelocation

textinput = "n";

sendBytes = Encoding.ASCII.GetBytes(textinput.PadRight(1024));

udpClient.Send(sendBytes, sendBytes.GetLength(0)); //send the packet

}//end of the try

catch (Exception e)

{

Console.WriteLine("Error with the Server Name: {0}", e.ToString());

Console.WriteLine("Did you start the Server First ?");

}//end of the catch

try

{

//the IP Address.any allows any valid matching address for this machine to be used

//i.e. loopback, broadcast, IPv4, IPv6

IPEndPoint remoteIPEndPoint = new IPEndPoint(IPAddress.Any, 8009); //open port 8009 on this machine

udpClient2.Client.ReceiveTimeout = 500; //sets timeout to prevent the programming hanging if no reply is recieved

recieveBytes = udpClient2.Receive(ref remoteIPEndPoint);

returnData = Encoding.ASCII.GetString(recieveBytes);

}

catch (Exception ex)

{

Console.WriteLine("Packet Timed out");

}

//grabs the counter value from the returned chunk packet. it only needs one value as the clients know to increment by 100000 immediately

int counter = 0;

try

{

counter = Convert.ToInt32(returnData);

}

catch

{

counter = 0;

Console.ReadLine();

Environment.Exit(0);

}

Console.WriteLine("Recieved Chunk {0} - {1}", counter, counter + 100000); //included to provide visual indication that the program is recieving chunks

String result = checkHash(hash, counter, counter + 100000); //pass to the check hash function

resultYN = result.Split()[0]; //the check hash function may pass back a yes result, this seperates the yes or no out for case checking

//if the result is positive, the client sends a result packet straight away, that contains a yes terminate for the server, and the actual hash value

if (resultYN == "y")

{

try

{

IPAddress remoteAddr = Dns.GetHostEntry(ServerName).AddressList[0]; //IP address of the server entered

udpClient.Connect(remoteAddr.ToString(), 8009); //address of the remotelocation

//read in the text from the console

textinput = result;

sendBytes = Encoding.ASCII.GetBytes(textinput.PadRight(1024));

udpClient.Send(sendBytes, sendBytes.GetLength(0)); //send the packet

}//end of the try

catch (Exception e)

{

Console.WriteLine("Error with the Server Name: {0}", e.ToString());

Console.WriteLine("Did you start the Server First ?");

}//end of the catch

}

}

//provides a delay to program close

Console.WriteLine("");

Console.ReadLine();

}

//checks the hashes against generated hashes

static string checkHash(string original, int start, int end)

{

string tocheck = "";

string foundHash = "n";

for (; start < end; start++)

{

tocheck = start.ToString();

if (original.CompareTo(generateHash(tocheck)) == 0)

{

Console.WriteLine("Found clear text is " + tocheck);

start = end;

foundHash = "y " + tocheck;

}

} //end of the FOR Loop

return foundHash;

}

//used to generate the hashes to check against

static string generateHash(string input)

{

//the method used here to generate the MD5 hash is a standard method provided by Microsoft

MD5 md5Hasher = MD5.Create();

byte[] data = md5Hasher.ComputeHash(Encoding.Default.GetBytes(input));

StringBuilder sBuilder = new StringBuilder();

for (int i = 0; i < data.Length; i++)

{

sBuilder.Append(data[i].ToString("x2"));

}

return sBuilder.ToString();

}

//This function runs in a thread, it constantly checks for a terminate signal

static void terminateThread()

{

//forces the client to join a multicast group listening on port 8012 for a global terminate signal, which will be sent once the correct hash has been found

UdpClient multicastClient = new UdpClient(8012);

IPAddress multicastIpAddress = IPAddress.Parse("225.0.0.1"); // assign the Multicast address

//join the Multicast group

multicastClient.JoinMulticastGroup(multicastIpAddress);

IPEndPoint remoteIPEndPoint = new IPEndPoint(IPAddress.Any, 8012);

Byte[] recieveBytes = multicastClient.Receive(ref remoteIPEndPoint);

string returnData = Encoding.ASCII.GetString(recieveBytes);

//Prints the values to the client machines then kills the thread

Console.WriteLine("The Hash has been Discovered: {0}", returnData);

Console.WriteLine("Program now ready to end!");

Thread.CurrentThread.Abort();

}

//provides initial connection to the server, and checks for timeout, if no response is found

static string serverConnect(string ServerName)

{

//creates updclient on this machine to recieve data

UdpClient udpClient = new UdpClient();

UdpClient udpClient2 = new UdpClient(8010);

Byte[] sendBytes = new Byte[1024]; // buffer to read the data into 1 kilobyte at a time

String textinput = null;

//requests input of ip of server - consider switching to a multicast to join the server group

String returnData = "";

String hello = null;

String hash = null;

//sends data to the server address, in this case a Hello packet, if a hello is recieved back then the loop ends or until 4 packets have been sent

int counter = 0;

while (counter < 4)

{

try

{

IPAddress remoteAddr = Dns.GetHostEntry(ServerName).AddressList[0]; //IP address of the server entered

udpClient.Connect(remoteAddr.ToString(), 8009); //address of the remotelocation

Console.WriteLine("Testing Connection");

//read in the text from the console

textinput = "Hello";

sendBytes = Encoding.ASCII.GetBytes(textinput.PadRight(1024));

udpClient.Send(sendBytes, sendBytes.GetLength(0)); //send the packet

}//end of the try

catch (Exception e)

{

Console.WriteLine("Error with the Server Name: {0}", e.ToString());

Console.WriteLine("Did you start the Server First ?");

}//end of the catch

try

{

Byte[] recieveBytes = new Byte[1024]; // buffer to read the data into 1 kilobyte at a time

//the IP Address.any allows any valid matching address for this machine to be used

//i.e. loopback, broadcast, IPv4, IPv6

IPEndPoint remoteIPEndPoint = new IPEndPoint(IPAddress.Any, 8010); //open port 8010 on this machine

udpClient2.Client.ReceiveTimeout = 500; //sets timeout to prevent the programming hanging if no reply is recieved

recieveBytes = udpClient2.Receive(ref remoteIPEndPoint);

returnData = Encoding.ASCII.GetString(recieveBytes);

hello = returnData.Split()[0];

hash = returnData.Split()[1];

}

catch (Exception ex)

{

Console.WriteLine("Packet {0} Timed out. Sending until 4!", counter + 1);

counter++;

}

if (counter == 4)

{

Console.WriteLine("Unable to establish connection: program now terminating!");

Console.WriteLine("Press enter to close");

}

if (hello == "Hello")

{

Console.WriteLine("Connected To Server!");

Console.WriteLine("");

counter = 4;

}

}

udpClient.Close();

udpClient2.Close();

return hash;

}

}

}

## Screenshots

### Server

### 

### 

### 

### 

### Client

### 

### 

## Discovered Issues

Due to the way the program terminates with the multicast, clients caught in a loop will hang for a while before finally exiting.

If a client disconnects just after a chunk has been received, there is no check for this and that chunk will be lost. If it contains the solution the program will fail.

The clients also need to have the IP address of the server manually entered, which is time consuming and could be automated through the use of multicasting a hello packet from a client, to receive a packet with the server address back.

# Recovered Clear Text

Given Hash: 9fe3d5324cf6fb6b4b472ace59f632fc

Decoded Value: 412,819,373

# Conclusions

This programming exercise has shown that through distributed computing it is more than possible to utilise CPU cycles on client machines to crunch complex problems in a faster fashion than performing the task on a single machine.

This program could be further adapted for more complex computations, such as longer length MD5 calculations and even alpha-numeric MD5 combinations.

While this program itself is not perfect as it can drop chunks if a client times out, it clearly demonstrates the amount of processing power available and the ease of which such a solution could be deployed.