Breaking RSA using MapReduce

Cloud Computing

Kristjan Jõgi
University of Tartu
Estonia, Tartu
linkin@ut.ee

Abstract — **This document describes my project in the Cloud Computing course.**

Keywords: RSA, integer factorization, MapReduce, Quadratic Sieve.

I. Introduction

The purpose of this document is to describe my project (including the progress and results of it) in the Cloud Computing course. The topic of the project is about breaking RSA using MapReduce.

II. Description of the project

The task of this project was to implement an integer factorization algorithm on MapReduce to break RSA public keys. The requirements for the algorithm were that it needs to be faster than trial division and it needs to be parallelizable. Below are some brief explanations of topics related to the project to better understand the purpose of it.

A. *RSA*

In cryptography, RSA (which stands for Rivest, Shamir and Adleman who first publicly described it) is an algorithm for public-key cryptography. It is the first algorithm known to be suitable for signing as well as encryption, and was one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be sufficiently secure given sufficiently long keys and the use of up-to-date implementations.

B. *Integer factorization*

In number theory, integer factorization or prime factorization is the breaking down of a composite number into smaller non-trivial divisors, which when multiplied together equal the original integer.

C. *Relationship between RSA and integer factorization*

The first steps of the RSA algorithm consist of choosing two distinct prime numbers p and q and then multiplying those – resulting in the product n which is also a part of the public key. The security of RSA relies on the fact that when the public key (particularly the integer n) is known to the attacker then retrieving p and q from it is computationally very complex and takes a lot of time. When speaking in the context of RSA, integer factorization attempts to find those prime numbers p and q when n is known. Due to the complexity of this process new and better integer factorization algorithms are being developed all the time.

III. Used technologies

I used various technologies to complete the project, they are listed down below.

A. *Java*

As a programming language I used Java which is a high-level programming language that runs in a virtual machine (JVM). It was also possible to use Python for the same task, but I prefer Java.

B. *Hadoop MapReduce*

MapReduce is a patented software framework introduced by Google in 2004 to support distributed computing on large data sets on clusters of computers. The framework is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as their original forms.

In my project I used the Hadoop MapReduce framework which is Apache's open-source implementation of MapReduce.

C. *Cygwin*

Cygwin is a Unix-like environment and command-line interface for Microsoft Windows. Cygwin provides native integration of Windows-based applications, data, and other system resources with applications, software tools, and data of the Unix-like environment.

It was necessary to use Cygwin, because as the operating system I use Windows which means that my testing environment for the project was Windows-based as well. The Hadoop MapReduce framework uses Unix command-line commands so Cygwin was needed to simulate the Unix environment.

D. *The quadratic sieve integer factorization algorithm*

The quadratic sieve algorithm (QS) is a modern integer factorization algorithm and, in practice, the second fastest method known (after the general number field sieve). It is still the fastest for integers under 100 decimal digits or so, and is considerably simpler than the number field sieve. It is a general-purpose factorization algorithm, meaning that its running time depends solely on the size of the integer to be factored, and not on special structure or properties. It was invented by Carl Pomerance in 1981 as an improvement to Schroeppel's linear sieve.

IV. Reason for choosing this project

The main reason for choosing this project was that in the combined course of Grid and Cloud Computing there were several assignments about breaking the RSA public key. In the Grid Computing part we had to write a parallelizable Python script to break RSA, and in the Cloud Computing part we had to do the same using Java. It's fair to say that being familiar with the topic played a big role.

Another reason was that the topic of integer factorization is also very interesting. It's interesting to implement algorithms to solve computationally complex problems.

V. Progress and problems

At first I started reading about integer factorization and available algorithms. The first algorithm I implemented was an optimized version of trial division, the optimization relied on some properties of prime numbers – so it also acted as a filter, resulting in less trial divisions – it was about 3.33 times faster than the normal version. However I concluded that this algorithm is not fast enough for the project.

Next I started looking into implementing some more complex integer factorization algorithms. The first one that I wanted to implement was the Quadratic Sieve factorization algorithm – however at first glance I found it to be too complex. After that I implemented the Shanks' Square Forms factorization algorithm – I realized it cannot be parallelized, because every iteration of it was dependant of the previous one.

Finally I still ended up implementing the Quadratic Sieve factorization algorithm. I used several single-threaded implementations as a reference, because implementing it from scratch using just the textual algorithm description is indeed very difficult and time consuming.

Currently, at this time, the project is fully finished and the final program can be successfully used to factorize integers. This project is probably one of the most complex and time-consuming projects I have ever completed.

VI. Basic idea of the Quadratic Sieve algorithm

- Input: integer *n*. Output: two prime factors (*p* and *q*) of *n*.
- It makes use of congruence of squares.
- It attemps to find two integers *x* and *y* so that $(x^2 \bmod n == y^2 \bmod n)$ and $(x \bmod n \mathrel{!=} y \bmod n)$.
- If such integers *x* and *y* are found then the two prime factors of *n* can be obtained by performing: $p = gcd(x - y, n)$ and $q = gcd(x + y, n)$. Here *gcd* stands for the greaters common divisor operation.
- Finding such integers *x* and *y* is difficult without doing any filtering.
- The algorithm uses a special sieving operation to filter out interesting *x* and *y* values from a large range.
- The resulting subset of *x* and *y* values is used to factorize *n*.

VII. Parallel version of the Quadratic Sieve on MapReduce

The parallel implementation of the Quadratic Sieve consists of three core objects – controller, mapper, reducer.

A. *The controller.*

- Calculates the total range for sieving.
- Creates subranges – with a length of 100000.
- Amount of ranges is dependant of the size of the input integer *n*. This approach is better than specifying the amount of ranges yourself – one CPU is enough to factorize very small integers.
- Writes subranges to the input file.
- Key – *LongWritable*, value – *Text*.
- *k* Map tasks get spawned. *k* is the number of subranges.

B. *The mapper.*

- One map task per sieving subrange.
- Gets key – value pairs from the controller.
- Each map task takes it's input range and performs the sieving operation, results in a sieved range.
- The sieved range is written to the output file.
- key – *LongWritable*, value – *Text*.
- When all map tasks are done, the reduce task starts.

C. *The reducer.*

- Gets key – value pairs from mappers.
- Only one reduce task, because sieving is slow, but the final factorization is fast.
- Tries to find *x* and *y* that were mentioned in the basic algorithm description and calculates the factors *p* and *q*.
- Writes the two prime factors of *n* to the output file.
- Key – *Text*, value – *Text*.
- Finally the controller reads the factors and displays them to the user.

VIII. Conclusion

This project was very interesting, it was certainly beneficial to further understand the basics of cryptographic systems including their security and vulnerabilities. However this project was also one of the most difficult and time-consuming projects that I have ever completed. The results of the project were successful.

References

[1] http://en.wikipedia.org/wiki/RSA

[2] http://en.wikipedia.org/wiki/Integer_factorization

[3] http://en.wikipedia.org/wiki/MapReduce

[4] http://en.wikipedia.org/wiki/Quadratic_sieve

[5] http://en.wikipedia.org/wiki/Cygwin