CSC-422: Project #2: Integer Factorization

Jon Thomas

2018-04-27

Introduction

This project uses Java with sockets to create a distributed network of machines that can factor a given integer that is a product of two primes. The security of several encryption algorithms depend on the difficulty of factoring, and this project aims to see how a distributed network of factoring machines perform better than a single machine with the same approach. This program uses Java's BigInteger class to allow for arbitrary integer lengths.

Algorithms

This program uses four different factoring algorithms to attempt factor an integer n that is a product of two primes, p and q. For the examples given, the '-g x' flag was used where the program would generate two x-bit primes and multiply them to create n. The program then runs the following algorithms:

Trial Division from 2 (TD2)

Trial division from 2 is the naive way of factoring and works well if n has small prime factors. It first checks if n is even, then iterates through all odd numbers until it finds a factor.

Trial Division from $\sqrt{n}$ (TDRN)

This is practically the same algorithm as TD2, except counting down from $\sqrt{n}$. The only real difference is that for every $i \in [2, \sqrt{n}]$, if $\gcd(n, i) > 1$, then it returns $\gcd(n, i)$.

Fermat's Factorization Method

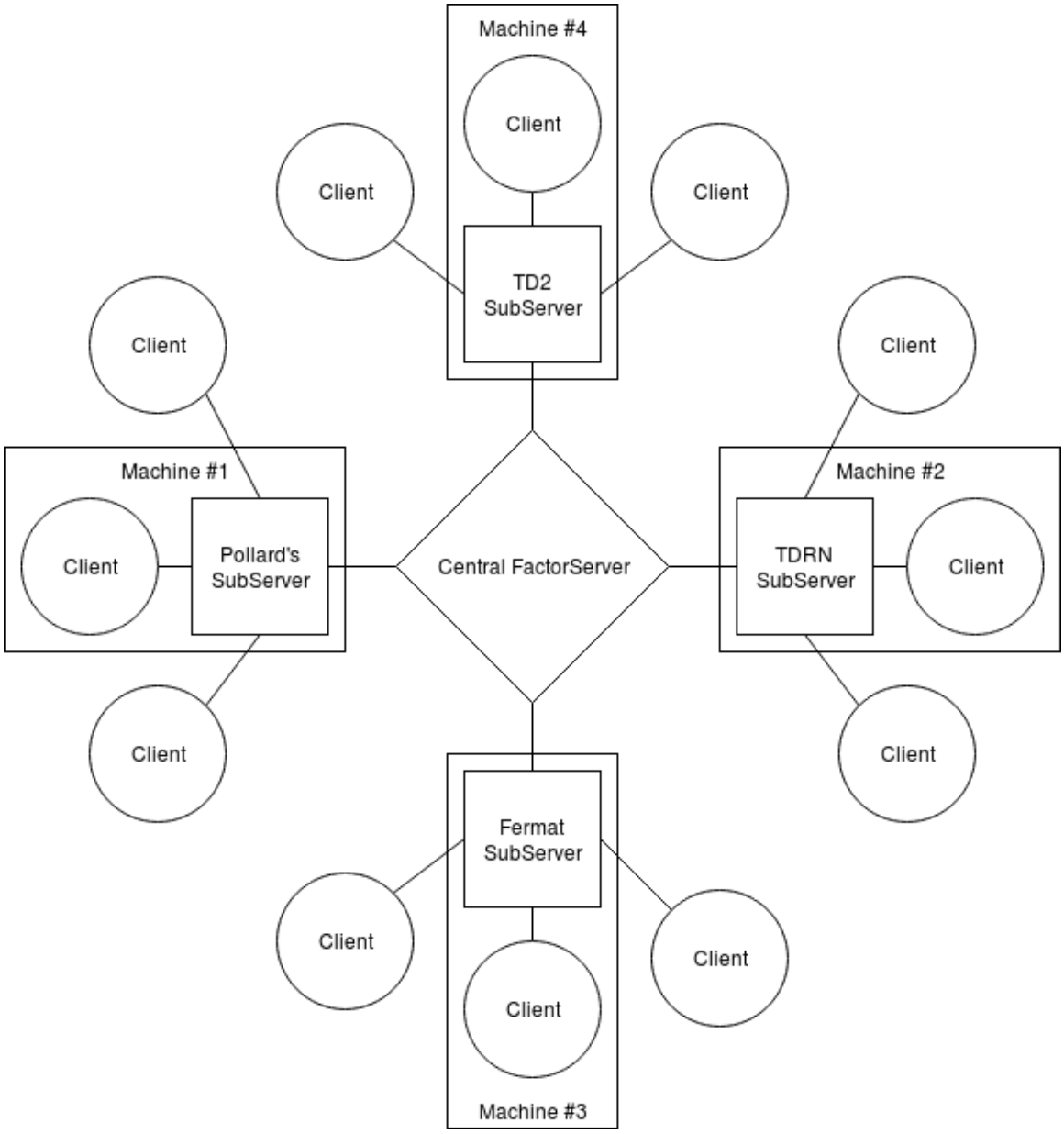
Attempts to find two integers a, b such that $a^2 - b^2 = n$. Then n has non-trivial factors of $(a + b)(a - b)$.

Pollard's p-1 algorithm

Try and find a value of x such that $x = a^{K(p-1)} \equiv 1 \mod p$, so that x is a multiple of a factor p of n. Then $\gcd(x - 1, n)$ will be a nontrivial factor of n. Particularly effective if $p - 1$ is comprised of a lot of small factors.

Networking

The networking works as follows:

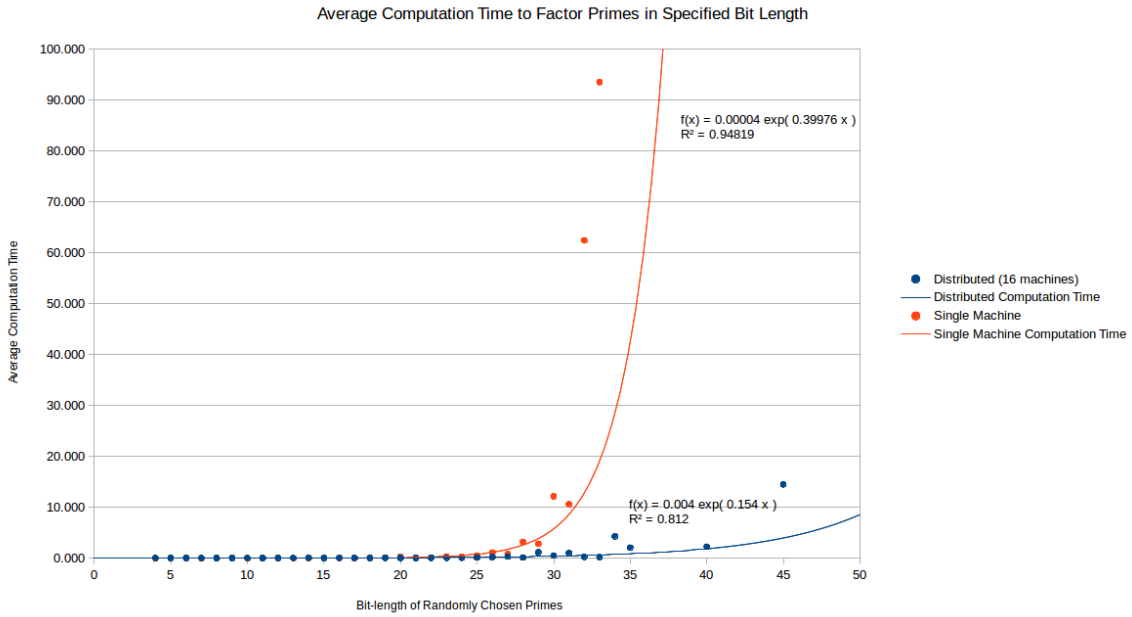


The network works by having a group of chosen sub-servers delegate clients for a specific algorithm. Each sub-server is in charge of a specific algorithm. The clients don't communicate with the main server except to initially connect. Instead, all client communication is done with these sub-servers. The program starts by launching the FactorServer. This server establishes a ServerSocket on a well-known port and waits for clients to connect with it. Clients are launched by specifying the machine on the network where the server is running. Clients will connect to the server's port where the main server will tell the client what machine is running its particular sub-server. If a client is one of the first four clients to connect to the server, then it is also responsible for launching a sub-server. These clients don't interact any differently with the sub-server despite running on the same machine. All future clients who connect will be given this sub-server's hostname to connect to. The sub-servers themselves also use well-known port numbers, but the machine's aren't known ahead of time. Delegation of which algorithms are assigned is done round-robin style, iterating through TD2,TDRN,Fermat and Pollard's. Each client is given a certain bound or restriction in its attempts to factor. If the client fails to succeed, the sub-server will give it a new bound to work in. This pattern continues until one client finds a factor. When a factor is found, the client sends a message to the sub-server who forwards it to the main server. The main server confirms the factor as valid, checks if any more factors exist, and if not will send a message to the sub-servers to quit.

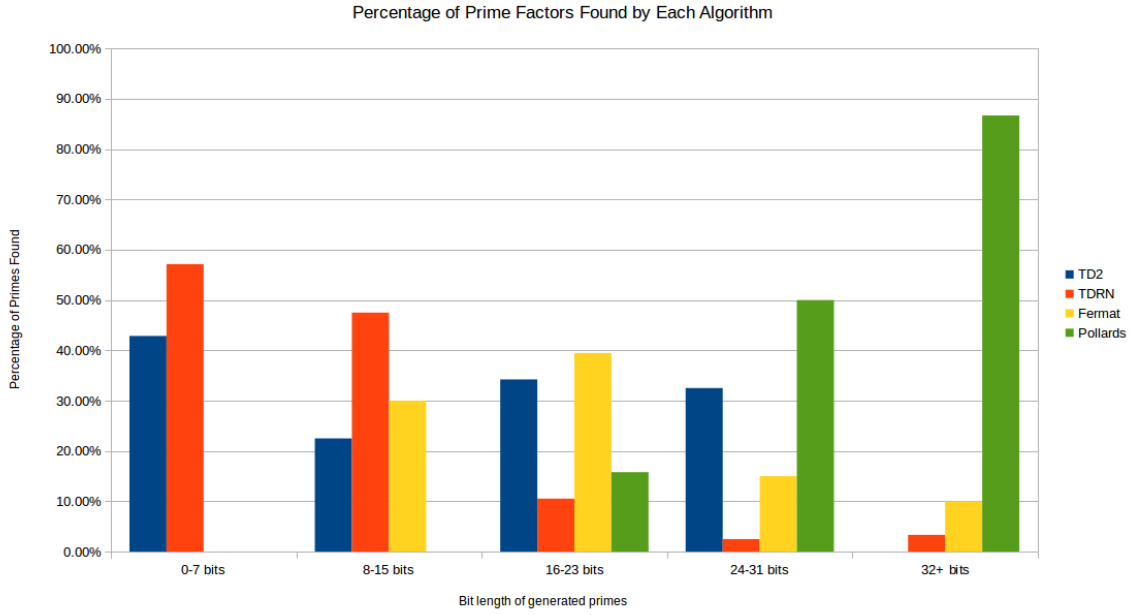
Timings

The efficiency of this program was tested against a single-machine implementation of the same algorithms. In the single-machine implementation (which is called FactorSequential, despite technically being parallel), each algorithm is launched in a seperate thread and runs over the whole bounds. For the distributed run, the program was tested over 17 machines (16 clients, 1 server). These machines were the Macs in the GS-930 computer lab on campus. This allowed 4 clients per algorithm to test that the algorithm was distributing properly. The following were the timing results from this experiment. Each timing run was testing for primes of a specified bit length (as longer primes would take more time to factor, and thus benefit from distributive programming). The following are an average of 5 runs per bit-length, each of which is randomly generated.

Prime Bit Length	Distributed Time (16 machines) (sec)	Single Machine Time (sec)
4	0.020	0.001
5	0.020	0.001
6	0.020	0.001
7	0.020	0.001
8	0.019	0.001
9	0.019	0.001
10	0.019	0.001
11	0.021	0.002
12	0.026	0.002
13	0.024	0.002
14	0.078	0.005
15	0.026	0.007
16	0.031	0.023
17	0.031	0.019
18	0.035	0.028
19	0.033	0.064
20	0.036	0.262
21	0.039	0.072
22	0.045	0.072
23	0.051	0.298
24	0.072	0.270
25	0.090	0.496
26	0.195	1.085
27	0.302	0.768
28	0.101	3.169
29	1.139	2.805
30	0.487	12.133
31	0.987	10.578
32	0.229	62.412
33	0.186	93.519
34	4.264	
35	2.064	
40	2.226	
45	14.502	



Interesting here is how much performance we get out of a distributed solution. On the single-machine solution, factoring starts to take longer than 1 second at around 26-27 bit primes. The distributed solution doesn't reliably exceed 1 second until around 34 bits, at which point the single-machine version has already exceed a minute and a half average time for factoring. Also to note is the greater difficulty of coming up with reliable timings for larger primes. This is likely due to properties had by some of the algorithms. Pollard's algorithm, for example, can factor quite quickly if n has a factor p where $p-1$ is comprised of small primes. Data about the success of the various algorithms can be seen below.



As can be seen, for small primes trial division is most effective. Fermat's Factoring Method starts to become more effective as the primes get larger, but Pollard's takes over as the only effective factoring method

above a certain point. It is important to note that Pollard's is effective when $p-1$ is comprised of only small primes. If $p-1$ has at least one large factor, then Pollard's has a running time potentially worse than trial division.

Conclusion

Distributed compuation is very effective for factoring, and is likely the only feasible way of factoring very large primes. The immediate performace boost on even small primes is noticeable, and single-machine factoring systems are not likely to be helpful for large projects. However, this program starts to notice significant slowdown after 40 or so bit primes. Extrapolating from the data recorded during the timing tests, 100 bit primes would take over an hour to factor and a 162 bit prime would take over a year. To factor a 1024 bit prime like those used in RSA would take these machines approximately 3.86×10^{57} years to factor, which is about 2.8×10^{47} times longer than the universe has existed. Thus, despite the obvious gains by distributing, using the Macs in the 930 computer lab to crack RSA encrypted messages would not be a feasible endeavor.