MAPREDUCE FOR INTEGER FACTORIZATION

JAVIER TORDABLE

ABSTRACT. Integer factorization is a very hard computational problem. Currently no efficient algorithm for integer factorization is publicly known. However, this is an important problem on which it relies the security of many real world cryptographic systems.

I present an implementation of a fast factorization algorithm on MapReduce. MapReduce is a programming model for high performance applications developed originally at Google. The quadratic sieve algorithm is split into the different MapReduce phases and compared against a standard implementation.

1. INTRODUCTION

The security of many cryptographic algorithms relies on the fact that factoring large integers is a very computationally intensive task. In particular RSA [1] would be vulnerable if there was an efficient algorithm to factor semiprimes (products of two primes). This could have severe consequences, as RSA is one of the most widely used algorithms in electronic commerce applications [2].

There are many algorithms for integer factorization [3]. From the trivial trial division to the classical Fermat's factorization method [4] and Euler's factoring method [5] to the modern algorithms, the quadratic sieve [6] and the number field sieve [7]. In particular the number field sieve algorithm was used in 1996 to factor a 512 bit integer [8], the lowest integer length used in commercial RSA implementations. There have been several other big integers factored over the course of the last decade. I would like to point out that in those cases the feat was accomplished with tremendous effort developing the software and a very considerable investment in hardware [9],[10].

In what follows I will expose how MapReduce, a distributed computational framework, can be used for integer factorization. As an example I will show an implementation of the quadratic sieve algorithm. I will also compare in terms of performance and cost a conventional implementation with the MapReduce implementation.

2. MAPREDUCE

I claim no participation in the development of the MapReduce framework. This section is basically a short extract of the original MapReduce paper by Jeff Dean and Sanjay Ghemawat [11]. MapReduce is a programming model inspired in computational programming. Users can specify two functions, *map* and *reduce*. The *map* function processes a series of (key, value) pairs, and outputs intermediate (key, value) pairs. The system automatically orders and groups all (key, value) pairs for a particular key, and passes them to the reduce function. The reduce function receives a series of values for a single key, and produces its output, which is sometimes a synthesis or aggregation of the intermediate values.

The canonical example of a MapReduce computation is the construction of an inverted index. Let's take a collection of documents $\mathcal{D} = \{D_0, D_1, ..., D_N\}$ which are composed of words $D_0 = (d_{0,0}, d_{0,1}, ..., d_{0,L_0}), D_1 = (d_{1,0}, d_{1,1}, ..., d_{1,L_1})$ and so on. We define a map function the following way:

$$map : (i, D_i) \rightarrow \{(d_{i,0}, (i, 0)), (d_{i,1}, (i, 1)), ..., (d_{i,L_i}, (i, L_i))\}$$

that is, for a given document it processes each word in the document and outputs an intermediate pair. The key is the word itself, and the value is the location in the corpus, indicated as (document, position). The reduce function is defined as:

$$reduce : \{(d, (i_1, j_1)), ..., (d, (i_L, j_L))\} \rightarrow (d, \{(i_1, j_1), ..., (i_L, j_L)\})$$

For a collection of pairs with the same key (the same word), it outputs a new pair, in which the key is the same, and the value is the aggregation of the intermediate values. In this case, the set of locations (document and position in the document) in which the word can be found in the corpus.

The MapReduce implementation automatically takes care of the parallel execution in a distributed system, data transmission, fault tolerance, load balancing and many other aspects of a high performance parallel computation. The MapReduce model escales seamlessly to thousands of machines. It is used continously for a multitude of real world applications, from machine learning to graph computations. And most importantly the effort required to develop a high performance parallel application with MapReduce is much lower than using other models, like for example MPI [12].

3. Quadratic Sieve

The Quadratic Sieve algorithm was conceived by Carl Pomerance in 1981. A detailed explanation of the algorithm can be found in [13]. Here we will just review the basic steps. Let N be the integer that we are trying to factor. We will attempt to find a, b such that: $N \mid (a^2 - b^2) \Rightarrow N \mid (a+b)(a-b)$. If $\{(a+b, N), (a-b, N)\} \neq \{1, N\}$ then we will have a factorization of N.

Lets define:

$$Q(x) = x^2 - N$$

if we find $x_1, x_2, ...x_K$ such that $\prod_{i=1}^{K} Q(x_i)$ is a perfect square, then:

$$N \mid \prod_{i=1}^{K} Q(x_i) - \left(\prod_{i=1}^{K} x_i\right)^2 = \prod_{i=1}^{K} \left(x_i^2 - N\right) - x_1^2 x_2^2 ... x_K^2$$

3.1. **Finding Squares.** Let's take a set of integers $x_1, ..., x_L$ which are B-smooth (all x_i factor completely into primes $\leq B$). One way to look for $i_1, i_2, ..., i_M$ such that $\prod_{j=1}^{M} x_{i_j}$ is a square is as follows. Let's denote p_i the i-th prime number. $\prod_{j=1}^{M} x_{i_j} = p_{j_1}^{a_1} p_{j_2}^{a_2} ... p_{j_L}^{a_L}$ is a square if and only if $2 \mid a_k$ for all $k \Leftrightarrow a_k \equiv 0 \, mod \, (2)$. For each x_i we will obtain a vector $v^i = v(x_i)$ where $v_j^i = max\left\{k : p_j^k \mid x_i\right\} \, mod \, (2)$. That is, each component j of v^i is the exponent of p_j in the factorization of x_i modulo 2. For example, for $B = 4$:

$$\begin{aligned} x_1 &= 6, v^1 = (1,1,0,0) \\ x_2 &= 45, v^2 = (0,0,1,0) \\ x_3 &= 75, v^3 = (0,1,0,0) \end{aligned}$$

It is immediate that:

$$v\left(\prod_{j=1}^{M} x_{i_j}\right) = \sum_{j=1}^{M} v\left(x_{i_j}\right)$$

Then

$$\prod_{j=1}^{M} x_{i_j} \text{is a square} \Leftrightarrow v\left(\prod_{j=1}^{M} x_{i_j}\right) = \overrightarrow{0}$$

In conclussion, in order to find a subset of $x_1, ..., x_L$ which is a perfect square, we just need to solve the linear system:

$$\left(\begin{array}{ccccccc} v^1 & | & v^2 & | & \dots & | & v^L \end{array} \right) \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_L \end{pmatrix} \equiv \overrightarrow{0}\, mod\,(2)$$

3.2. **Sieving for smooth numbers.** Back to the original problem, we just need to find a convenient set $\{x_1, x_2, ..., x_L\}$ such that $\{Q(x_1), Q(x_2), ..., Q(x_L)\}$ are B-smooth numbers for a particular B. First of all, lets notice that we don't need to consider every prime number $\leq B$. If a prime p verifies: $p \mid Q(x)$ for some x then:

$$p \mid Q(x) \Leftrightarrow p \mid x^2 - N \Leftrightarrow x^2 \equiv N\, mod\,(p) \Leftrightarrow \left(\frac{N}{p}\right) = 1$$

Because N is a quadratic residue modulo p if and only if the Legendre symbol of n over p is 1. We will take a set of primes which verifies that property and we will call it *factor base.*

In order to consider smaller values of $Q(x)$ we will take values of x around $\sqrt{N}$, i.e. $x \in \left[\lfloor\sqrt{N}\rfloor - M, \lfloor\sqrt{N}\rfloor + M\right]$ for some M. Both B above and M here are chosen as indicated in [13].

In order to factor all the $Q(x_i)$ we will use a method called *sieving* which is what gives the quadratic sieve its name. Notice that $p \mid Q(x) \Rightarrow p \mid Q(x+kp) = x^2 + 2kpx + k^2p^2 - N = \left(x^2 - N\right) + p\left(2kx + k^2p\right)$. Then

$$Q(x) \equiv 0\, mod\,(p) \Rightarrow \forall k \in \mathbb{N}, Q(x+kp) \equiv 0\, mod\,(p)$$

We can solve the equation $Q(x) \equiv 0\, mod\,(p) \Leftrightarrow x^2 - N \equiv 0\, mod\,(p)$ efficiently and obtain two solutions s_1, s_2 [14]. If we take:

$$z_{p,\{1,2\}} = min\left\{x \in \left[\lfloor\sqrt{N}\rfloor - M, \lfloor\sqrt{N}\rfloor + M\right] : x \equiv s_{\{1,2\}}\, mod\,(p)\right\}$$

then all $Q\left(z_{p,\{1,2\}} + kp\right), k \in [0, K]$ are divisible by p. We can divide each one of them by the highest power of p possible. For example:

$$
\begin{aligned}
(x_i) = & \quad (\ldots, 6, 7, 8, 9, 10, \ldots) \\
(Q(x_i)) = & \quad (\ldots, -41, -28, -13, 4, 23, \ldots) \\
& \left(\tfrac{77}{2}\right) = 1 \text{ as } 77 \equiv 1 \equiv 1^2 \, mod\,(2) \\
& x^2 - 77 \equiv 0 \, mod\,(2) \text{ yields } 1, 3, 5, 7, 9, ... \\
& (\ldots, -41, -7, -13, 1, 23, \ldots) \qquad \text{after sieving by 2}
\end{aligned}
$$

After sieving for every appropriate p, all the $Q(z)$ that are equal to 1 are smooth over the factor base.

4. Method

I developed a basic implementation of the Quadratic Sieve MapReduce which runs on Hadoop [15]. Hadoop is an open source implementation of the MapReduce framework. It is made in Java and it has been used effectively in configurations ranging from one to a few thousand computers. It is also available as a commercial cloud service [16].

This implementation is simply a proof of concept. It relies too heavily on the MapReduce framework and it is severy bound by IO. However the size and complexity of the implementation are several orders of manitude lower than many competing alternatives.

The 3 parts of the program are :

- *Controller*: Is the master job executed by the platform. It runs before spawning any worker job. It has two basic functions: first it generates the factor base. The factor base is serialized and passed to the workers as a counter. Second it generates the full interval to sieve. All the data is stored in a single file in the distributed Hadoop file system [17]. It then relies on the MapReduce framework to automatically split it in an adequate number of shards and distribute it to the workers
- *Mapper*: The mappers perform the sieve. Each one of them receives an interval to sieve, and they return a subset of the elements in that input sieve which are smooth over the factor base. All output elements of all mappers share the same key
- *Reducer*: The reducer receives the set of smooth numbers and attempts to find a subset of them whose product is a square by solving the system modulo 2 using direct bit manipulation. If it finds a suitable subset, it tries to factor the original number, N. In general there will be many subsets to choose from. In case that the factorization is not succesful with one of them, it proceeds to use another one. The single output is the factorization

In order to compare performance I developed another implementations of the Quadratic Sieve algorithm in Maple. They are both available online[1]. Both implementations are basic in the sense that they implement the basic algorithm described above and the code has not been heavily optimized for performance. There are many differences between the two frameworks used that could impact performance. Because of that a direct comparison of running times or memory space may not be meaningful. However it is interesting to notice how each of the implementations scales depending on the size of the problem.

[1] http://www.javiertordable.com/research.html TBD

Decimal	Sieve	MapReduce		Maple	
Digits	Size	Time (s)	Memory (MB)	Time (s)	Memory (MB)
10	5832	2.0	149.6	0.1	7.5
15	85184	3.0	397.1	3.5	15.5
20	970299	35.0	463.1	116.0	100.8
25	7529536	495.0	670.0	3413.7	894.0

TABLE 1. Absolute performance of the MapReduce and Maple implementations

Decimal	Sieve	MapReduce		Maple	
Digits	Size	Time	Memory	Time	Memory
10	1.0	1.0	1.0	1.0	1.0
15	14.6	1.5	2.7	35.0	2.1
20	166.4	17.5	3.1	1160.0	13.4
25	1291.1	247.5	4.5	34137.0	119.2

TABLE 2. Normalized performance of the MapReduce and Maple implementations

Decimal	Absolute Sieve	Relative Sieve	Absolute	Relative
Digits	Size	Size	Disk (MB)	Disk (MB)
10	5832	1.0	0.1	1.0
15	85184	14.6	2.1	14.6
20	970299	166.4	29.4	166.4
25	7529536	1291.1	275.3	1291.1

TABLE 3. Disk usage of the MapReduce implementation

5. RESULTS

Figures 1 and 2 show the results both in absolute terms and normalized. Figure 3 shows the disk usage of the MapReduce implementation. To test both implementations I took a set of numbers of different sizes[2]. The number of decimal digits d is indicated in the first column of each table. In order to contruct those numbers I took two factors close to $10^{\frac{d}{2}}$, with their product slightly over 10^d.

In each table sieve size indicates the number of elements that the algorithm analyzed in the sieve phase. For the MapReduce application the time result is taken from the logs, and the memory result is obtained as the maximum memory used by the process. For the Maple implementation both time and memory data are taken from the on screen information in the Maple environment. Finally disk usage data for the MapReduce is taken as the size of the file that contains the list of numbers to sieve. The Maple program runs completely in memory for the samples analyzed.

[2] 1164656837, 117375210056563, 10446257742110057983, 1100472550655106750000029

6. Discussion

The MapReduce implementation has a relatively big setup cost in time and memory when compared with an application in a conventional mathematical environment. However it scales better with respect to the size of the input data.

MapReduce is optimized to split and distribute data form disk. If an application handles a significant volume of data, IO capacity and performance can be a limiting factor. In our case disk usage is directly proportional to the size of the sieve set, which grows exponentially on the number of digits.

Both MapReduce and Maple implementations are similar in terms of development effort. The Maple implementation seems more adequate for small-sized problems while the MapReduce application is more efficient for medium-sized problems. Also it will be easier to scale in order to solve harder problems.

References

[1] Rivest, R.; A. Shamir; L. Adleman. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21 (2): 120–126.

[2] Nash, A., Duane, W., and Joseph, C. 2001. Pki: Implementing and Managing E-Security. McGraw-Hill, Inc.

[3] Donald Knuth. 1997. The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Third Edition. Addison-Wesley. ISBN 0-201-89684-2. Section 4.5.4: Factoring into Primes, pp. 379–417

[4] Israel Kleiner. 2005. Fermat: The Founder of Modern Number Theory. Mathematics Magazine, Vol. 78, No. 1 (Feb., 2005), pp. 3-14

[5] McKee, James. 1996. Turning Euler's Factoring Method into a Factoring Algorithm"; in Bulletin of the London Mathematical Society; issue 28 (volume 4); pp. 351-355

[6] Pomerance, C. 1985. The quadratic sieve factoring algorithm. In Proc. of the EUROCRYPT 84 Workshop on Advances in Cryptology: theory and Application of Cryptographic Techniques. Springer-Verlag New York. 169-182.

[7] Lenstra, A. K., Lenstra, H. W., Manasse, M. S., and Pollard, J. M. 1990. The number field sieve. In Proceedings of the Twenty-Second Annual ACM Symposium on theory of Computing. ACM, New York, NY, 564-572.

[8] Cowie, J., Dodson, B., et al. 1996. A World Wide Number Field Sieve Factoring Record: On to 512 Bits. In Proceedings of the international Conference on the theory and Applications of Cryptology and information Security. Lecture Notes In Computer Science, vol. 1163. Springer-Verlag, London, 382-394.

[9] Golliver, R. A., Lenstra, A. K., and McCurley, K. S. 1994. Lattice sieving and trial division. In Proceedings of the First international Symposium on Algorithmic Number theory. L. M. Adleman and M. A. Huang, Eds. Lecture Notes In Computer Science, vol. 877. Springer-Verlag, London, 18-27.

[10] S. Cavallar and W. M. Lioen and H. J. J. Te Riele and B. Dodson and A. K. Lenstra and P. L. Montgomery and B. Murphy Et Al and Mathematisch Centrum. 2000. Factorization of a 512-bit RSA modulus. Proceedings of Eurocrypt 2000. Springer-Verlag. 1-18.

[11] Dean, J. and Ghemawat, S. 2004. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004, 137-150

[12] Gropp, W., Lusk, E., and Skjellum, A. 1994. Using Mpi: Portable Parallel Programming with the Message-Passing Interface. MIT Press. 257-260

[13] Carl Pomerance. 1996. A Tale of Two Sieves, Notices of the AMS, 1473-1485

[14] Niven, I. and Zuckerman, H.S. and Montgomery, H.L. 1960. An introduction to the theory of numbers. John Wiley and Sons, Inc. 110-115

[15] http://hadoop.apache.org/

[16] http://aws.amazon.com/elasticmapreduce/

[17] Borthakur, D. 2007. The hadoop distributed file system: Architecture and design. http://svn.apache.org/repos/asf/hadoop/core/tags/release-0.15.3/docs/hdfs_design.pdf