Web Algorithms

Jason Hunter

Principal Technlogist
Mark Logic
http://marklogic.com

Session TS-6045

Overarching Goal

Understand a few of the classic algorithms we use everyday on the Web, whether we realize it or not

The Algorithms

XOR Swap

Credit Card Validation

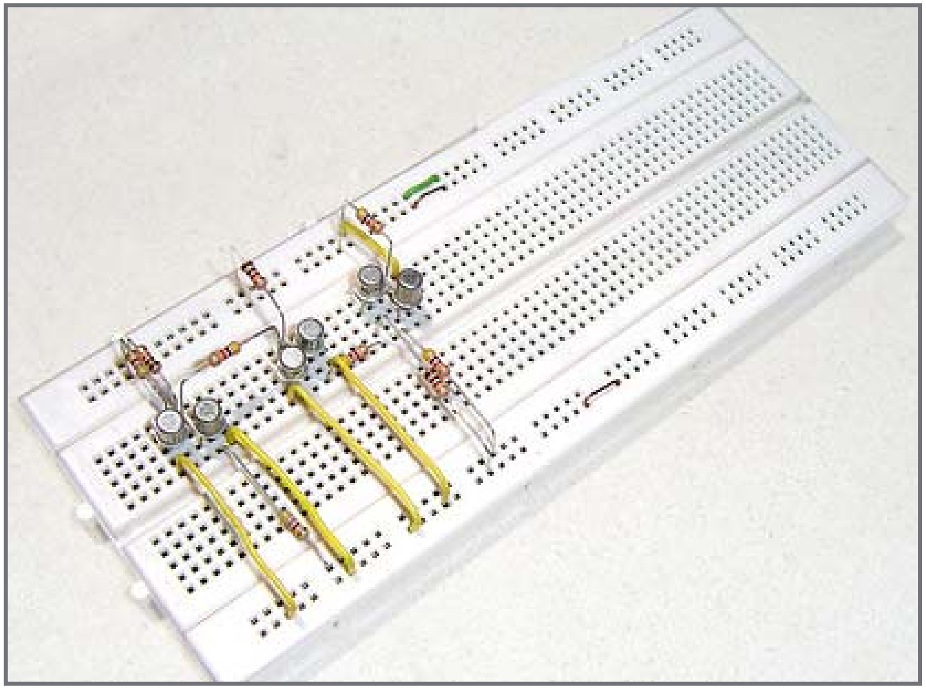
Public Key Cryptography

Two’s Complement

Google MapReduce

Puzzle #1

- How do you swap two variable values without using a temporary variable?



http://www.flickr.com/photos/curveto/157107227/

Answer

- You can use the XOR Swap algorithm:

```
x = x xor y
y = x xor y
x = x xor y
```

- XOR is a bitwise “exclusive or”

XOR Swap in the Java™ Programming Language

- XOR is performed in the Java programming language using the carat

```
public class Swap {
  public static void main(String[] args) {
    int x = 34, y = 78;
    x = x ^ y;    // or x ^= y;
    y = x ^ y;    // or y ^= x;
    x = x ^ y;    // or x ^= y;
    System.out.println(x + " " + y);
  }
}
```

XOR Swap in the Java Programming Language

- Here’s what happens at the bit level

```
int x = 34;   // 0b00100010
int y = 78;   // 0b01001110
x = x ^ y;    // 0b01101100 (108)
y = x ^ y;    // 0b00100010 (34)
x = x ^ y;    // 0b01001110 (78)
```

- The “108” value is special because given X you can find Y, and given Y you can find X
- We use that to pull the original values out

XOR Swap in the Java Programming Language

- It's easier to read if you don't reuse names

```
int x = 34;  // 0b00100010
int y = 78;  // 0b01001110
a = x ^ y;   // 0b01101100 (108)
y2 = a ^ y;  // 0b00100010 (34)
x2 = a ^ x;  // 0b01001110 (78)
```

Should You?

- Cool. Should you actually use XOR swap?
  - No. Trust your compiler. Your compiler probably won't use XOR as it forces serialized execution
  - Plus you need to watch for aliasing on languages that support it (not Java programming language)

Add/Subtract Swap?

- Can you craft a version of the XOR Swap using just addition and subtraction?

Add/Subtract Swap

- You sure can

```
x = x + y
y = x - y
x = x - y

int x = 34;
int y = 78;
x = x + y;   // 112
y = x - y;   // 34
x = x - y;   // 78
```

- But you do have to worry about overflow!

The Algorithms

XOR Swap

Credit Card Validation

Public Key Cryptography

Two's Complement

Google MapReduce

Puzzle #2

- How can you pre-check a credit card number on a web form?

Answer

- You can use the Luhn algorithm
  - A system that adds a “check digit” on the end of a number sequence
  - Used by most credit cards as well as Canadian social insurance numbers
  - Useful to catch errors quickly
  - Ideal for client-side JavaScript™ programming language

The Luhn Algorithm

- To check if a number passes the checksum
  - Start with the right-most number (checksum)
  - Move left, doubling every second digit
  - For any digits over 10 add their digits, so 6*2 = 12 becomes 3
  - Sum the generated digits
  - If the sum ends in 0, it’s valid. If not, invalid.

An Example

←

446-667-651

Digit	Double	Sum
1		1
5	10	1
6		6
7	14	5
6		6
6	12	3
6		6
4	8	8
4		4
		40

JavaOne

Why Double?

- The doubling is designed to catch transpositions
  - 1234 = 4+6+2+2 = 14
  - 1243 = 3+8+2+2 = 15
- Hmm, 90 and 09 transpositions aren't caught
  - 1290 = 0+(18=9)+2+2 = 13
  - 1209 = 9+0+2+2 = 13

Why So Simple?

- It was intended for a mechanical device
  - Explained in Patent 2,950,048, titled "Computer for Verifying Numbers"
  - Filed in 1954, granted in 1960
  - (Long ago expired)

Patent 2,950,048

United States Patent Office

2,950,048
Patented Aug. 23, 1960

1

2,950,048

COMPUTER FOR VERIFYING NUMBERS

Hans P. Luhn, Armonk, N.Y., assignor to International Business Machines Corporation, New York, N.Y., a corporation of New York

Filed Jan. 6, 1954, Ser. No. 402,491

5 Claims. (Cl. 235—61)

This invention relates to a hand computer for computing a check digit for numbers or for verifying numbers which already have a check digit appended.

The principal object of the invention is to provide a simple, inexpensive and portable computer for computing check digits and to provide a simple device for verifying numbers which have a check digit appended.

A further object of the invention is to provide apparatus for computing, in a fast and simple manner, check digits to append to the numbers or to verify numbers with check digits attached.

Pursuant to the invention, a visual check is provided for use at the time of verification. Stamping means is also preferably provided for recording the verified number and for preserving the visual check, which may be appended to the number.

The apparatus of my invention is used in a checking system for multi-digit numbers to indicate whether, in transmitting a number, an error has been made, such as a transposition of the digits. It may be used, for example, where a great many parts are ordered, manufactured, invoiced, shipped, and billed by multi-digit numbers. When a number is first assigned to a new part a check digit is computed, as will be explained hereinafter, and this check digit is appended to the righthand end of the part number. Thereafter whenever the correctness of that part number is in question the number can always be easily and quickly verified by my invention.

The particular mathematical system of number checking preferably embodied in my invention is one in which a single digit, called the check digit, is appended to the righthand end of the original or true number. The value of this check digit is so computed that in verifying the number by cross addition of the multiple digits of the number and the check digit, in accordance with a rule of substitution, the result will be a zero. This zero will appear as such on the computer. If the stamping or printing means of my device is utilized, a check mark may be used to indicate that the number is correct.

Specific illustrations of my invention are shown in the accompanying drawings illustrating two embodiments of the invention, and in which:

Figure 1 is a front perspective view of one of the said embodiments of my device;

Figure 2 is a front elevation of one of the said embodiments partially in cross section;

Figure 3 is a cross-section of one of the said embodiments on the line 3—3 in Figure 2;

Figure 4 is a perspective view of a portion of the same, partly in cross section;

Figure 5 is a front elevation of another embodiment of my invention; and

Figure 6 is a sectional view taken on the line 6—6 in Figure 5.

For convenience of description, the operation of the apparatus of my invention, first in computing a check digit and secondly in verifying a number with a check digit appended, will be set forth to facilitate a complete

2

understanding of the function and purpose of the apparatus. This will be followed by a description of the apparatus and its operation.

It is commonly known that in copying a number comprised of a plurality of digits it often happens that an error occurs by transposing two of the digits. This common error is detected by the invention herein described by the cross addition of digits, the alternate digits being replaced by "substitute" digits, prior to the cross addition. It should be understood that other systems of cross addition checking could be utilized but the system used herein is described as a practical example. In such a method of cross addition for checking a number, it is readily seen that the straight cross addition of the original digits of a number would fail to give any information concerning erroneous transposition because the sum would be the same regardless of the relative placement of the digits. However, if every other digit is a substitute digit in accordance with the system herein set forth, such an error will be detected.

The substitute digit equals twice the origianl digit plus an end around carry (an end around carry in this system means the addition of any digit standing in the tens position to the digit standing in the units position in the doubled number, as shown below). Thus the substitute digit for an original –3– is –6–(–2–×–3–=–6–). The substitute digit for an original –6–, illustrating the end around carry, is –3–(–2–×–6–=–12–=–1–+–2–=–3–).

The following table gives the substitute for each digit according to this system.

Original	–0–	–1–	–2–	–3–	–4–	–5–	–6–	–7–	–8–	–9–
Substitute	–0–	–2–	–4–	–6–	–8–	–1–	–3–	–5–	–7–	–9–

Applying this system of substitute digits to determine the check digit for a number of seven digits (which is the number of digits provided for in the particluar embodiment of the invention hereinafter described), such as –4872148–, first a check digit will be determined, and secondly the number with the check digit appended will be verified. In accordance with the substitution system utilized in my invention, the first digit of the number reading from left to right is a substitute digit, the second digit is an original digit and then this order is repeated until all of the digits have been accounted for. The first digit of the example number, the original –4–, is replaced by its substitute digit, an –8–. This –8– is added to the next digit –8–, an original digit, resulting in the sum of –16– which becomes a –6– by casting out tens in the usual manner. The next digit is an original –7– which is replaced by its substitute digit, a –5–. This –5– is added to the –6– resulting in a –1–. This cross addition, if continued in accordance with the above, across the remaining four digits of the sample number would result in a sum of –6–. This can be determined from the following table giving the original and alternate substitute digits for the number in question.

Original ---------------------- –4 8 7 2 1 4 8–
Alternate Substitute -- –8+8+5+2+2+4+7– = –6–

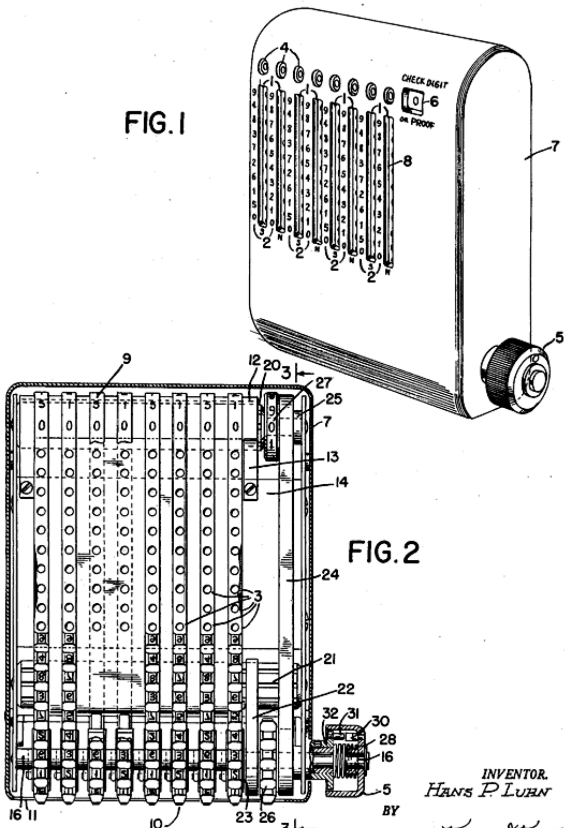
Once this sum of –6– has been computed the check digit to be appended is derived by adding to this sum its tens complement or in this case the digit –4–, this being the amount to be added to –6– to produce ten. If this –4– is added to the sum –6– as an original number, the total in the last column will be a –0–. The significance of this particular end result will become apparent in the explanation of the verification of a number having a check digit appended.

It should be realized that the check digit should be added in as an original number. This is accomplished by starting out by using either the original or the substitute digit for the first digit of the alternate substitute

Aug. 23, 1960 H. P. LUHN 2,950,048

COMPUTER FOR VERIFYING NUMBERS

Filed Jan. 6, 1954 3 Sheets-Sheet 1



INVENTOR.
HANS P. LUHN
BY
Kenyon & Kenyon
ATTORNEYS

Luhn Check

```
public static boolean checkNumber(int[] digits) {
  int sum = 0;
  boolean alt = false;
  for (int i = digits.length - 1; i >= 0; i--) {
    if (alt) {
      int doubled = digits[i] * 2;
      if (doubled > 9) {
        doubled -= 9;  // equiv to adding digits
      }
      sum += doubled;
    }
    else {
      sum += digits[i];
    }
    alt = !alt;
  }
  return sum % 10 == 0;
}
```

Luhn Create

```
public static int createChecksum(int[] digits) {
  int sum = 0;
  boolean alt = true;
  for (int i = digits.length - 1; i >= 0; i--) {
    if (alt) {
      int doubled = digits[i] * 2;
      if (doubled > 9) {
        doubled -= 9;  // equiv to adding digits
      }
      sum += doubled;
    }
    else {
      sum += digits[i];
    }
    alt = !alt;
  }
  return (10 - (sum % 10)) % 10;
}
```

Luhn Check in JavaScript Programming Language

```
function checkNumber(number) {
  if (/\D/.test(number)) return false;
  digits = (number+'').split('');
  var sum = 0; alt = false;
  for (var i = digits.length - 1; i >= 0; i--) {
    if (alt) {
      doubled = parseInt(digits[i]) * 2;
      if (doubled > 9) {
        doubled -= 9;  // equiv to adding digits
      }
      sum += doubled;
    }
    else {
      sum += parseInt(digits[i]);
    }
    alt = !alt;
  }
  return sum % 10 == 0;
}
```

Other Checks

- Can you do other checks? Yes
  - Visa cards begin with 4
  - MasterCard cards begin with 51 thru 55
  - AmEx cards begin with 34 or 37
  - Discover cards begin with 6011
- Is it better to ask for type or recognize type?

The Algorithms

XOR Swap

Credit Card Validation

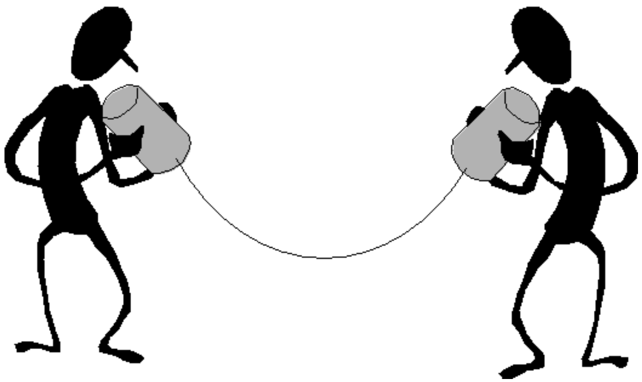
Public Key Cryptography

Two's Complement

Google MapReduce

Puzzle #3

- How can you communicate securely over a public network without pre-agreeing on a shared secret?



Answer

- Using Public Key Cryptography
  - The underpinnings of protocols like https and SSH®
- I'll give an overview, then look at RSA

Symmetry

- Symmetric encryption
  - Use the same key to encrypt and decrypt
  - DES and 3DES, Blowfish, AES (Rijndael)
- Asymmetric (public key) encryption
  - One key to encrypt, another to decrypt
  - RSA, ElGamal, DSA

Public Keys

- With asymmetric keys, I have one as public, one as private
  - I keep the private safe, maybe encrypted with a symmetric key
  - I share the public far and wide
  - You can encrypt messages with my public key, and only I can decrypt them

Trusting a Public Key

- How can you trust you have my genuine public key?
  - Talk to me and I'll confirm its fingerprint
  - Or it could be signed by a Certificate Authority (VeriSign, Thawte)
  - Or it could be signed by someone you trust ("web of trust")

Signing

- How do I trust the message you sent was truly from you?
  - You encrypt the message with your private key as well as my public key
  - I decrypt with your public, then my private
  - Provides “authenticity” and “confidentiality”

Performance

- Asymmetric algorithms are slower than symmetric
  - So you don't really encrypt whole messages
  - You encrypt with a symmetric key and pass the key using asymmetric keys
  - To prove authorship you hash the message and encrypt the hash
  - That's why you see: RSA, 3DES, SHA1

Uses

- Public Key Cryptography has lots of uses
  - Prevent eavesdropping, tampering, and impersonation
  - Secure communication on untrusted networks (https, SSH®, encrypted email)
  - Electronic signatures
  - Digital cash

RSA

- The concept of public key crypto was invented by Diffie, Hellman, and Markle
- The marquee implementation was invented in 1977 by Rivest, Shamir, and Adleman at MIT
- Patent 4,405,829 (now expired)
- http://www.ladlass.com/intel/archives/010256.html

The Basic Math

- Choose two large primes, “p” and “q”
- Multiply to produce a product, “n”
- (It’s believed hard to calculate p and q given just n if n is large, ~2048 bits or higher)

The Basic Math

- Choose an encryption component, “e”
  - Often 65537 (216 + 1)
- Calculate a decryption component, “d”
  - To calculate “d” you need “e”, “p”, and “q”
  - d*e mod (p-1)(q-1) must be 1
  - d = e-1 mod (p-1)(q-1)

Encrypt/Decrypt

- C is the ciphertext, M is the message
  - $C = M^e \bmod n$
  - $M = C^d \bmod n$
- Proving this involves Fermat's little theorem and the Chinese remainder theorem

Example

- p = 61, q = 53
- n = 61*53 = 3233
- Choose e = 17
- Calculate d = 2753
  - Yes, 2753*17 mod 3120 = 1

Example

- Message is “123”
- $C = 123^{17} \bmod 3233 = 855$
- $M = 855^{2753} \bmod 3233 = 123$

- Knowing 17 and 3233 you can’t get 2753
- Knowing 2753 and 3233 you can’t get 17

Example in the Java Programming Language

```
// http://www.cs.princeton.edu/introcs/79crypto/RSA.java.html

public class RSA {
  private static BigInteger one = new BigInteger("1");
  private static SecureRandom random = new SecureRandom();
  private BigInteger privateKey, publicKey, modulus;

  // Generate an N-bit (roughly) public and private key
  RSA(int n) {
     BigInteger p = BigInteger.probablePrime(n/2, random);
     BigInteger q = BigInteger.probablePrime(n/2, random);
     BigInteger phi = (p.subtract(one)).multiply(q.subtract(one));
     modulus    = p.multiply(q);
     publicKey  = new BigInteger("65537");
     privateKey = publicKey.modInverse(phi);
  }
```

Example in the Java Programming Language

```
BigInteger encrypt(BigInteger message) {
  return message.modPow(publicKey, modulus);
}
BigInteger decrypt(BigInteger encrypted) {
  return encrypted.modPow(privateKey, modulus);
}

public static void main(String[] args) {
  int n = Integer.parseInt(args[0]);  // key size
  RSA key = new RSA(n);

  // create random message, encrypt and decrypt
  BigInteger message = new BigInteger(n-1, random);
  BigInteger encrypt = key.encrypt(message);
  BigInteger decrypt = key.decrypt(encrypt);
}
```

The Algorithms

XOR Swap

Credit Card Validation

Public Key Cryptography

Two's Complement

Google MapReduce

Puzzle #4

- Why does 2,000,000,000 + 2,000,000,000 equal -294,967,296?

Answer

- Because of overflow and the details involved with two’s complement notation

Integers

- Computers may represent integers in several different ways, including
  - Sign-and-magnitude
  - Ones' complement
  - Two's complement

Sign-and-Magnitude

- Sign-and-magnitude uses one bit to represent the sign and the remaining bits represent the magnitude (absolute value)
  - It's a lot like how humans write: +5, -5
  - Sign bit of 0 is positive, 1 is negative
  - Used on early binary computers (IBM 7090)
  - Has both positive and negative 0

Java

JavaOne

4-Bit Integers

Decimal	Sign and Magnitude
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
+0	0000
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111
-8	N/A

One's Complement

- One's complement represents negative numbers as the bitwise not of the positive
  - Still a sign bit, still two values of 0
  - Used by PDP-1 and Univac 1100/2200
  - Named for subtracting from a long string of ones (0b1111-0b0010 = 0b1101)
  - A "bitwise not"

4-Bit Integers

Decimal	Sign and Magnitude	Ones' Complement
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
-0	1000	1111
-1	1001	1110
-2	1010	1101
-3	1011	1100
-4	1100	1011
-5	1101	1010
-6	1110	1001
-7	1111	1000
-8	N/A	N/A

Two’s Complement

- Two’s Complement represents negative numbers as one’s complement plus one
  - Still a sign bit, no negative zero, one extra negative value
  - By far the most common today
  - Named for subtracting from 2^n (which is 10000000, or one larger than n-many ones)

4-Bit Integers

Decimal	Sign and Magnitude	Ones' Complement	Two's Complement
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	1000	1111	N/A
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	N/A	N/A	1000

Neat Trick

- You can convert a two’s complement number to decimal by adding its bits, assigning a negative value to the highest bit
  - 0b11111011 as an 8-bit number
    = -128+64+32+16+8+0+2+1
    = -5

Why So Common?

- Two’s Complement is ubiquitous because addition and subtraction operations can be unified, plus there’s no weird -0 value
  - Consider 3+1 and 3+(-1) using 4-bit numbers

```
 1111     (carry)          11      (carry)
0b0011 +      (3)       0b0011 +      (3)
0b1111       (-1)       0b0001        (1)
============            ============
0b0010        (2)       0b0100        (4)
```

(Note: You only include the last 4 bits)

Overflow

- Integer.MAX_VALUE is 2,147,483,647
- Integer.MIN_VALUE is -2,147,483,648
- Integer.MAX_VALUE + 1 = Integer.MIN_VALUE
- And 2,000,000,000 + 2,000,000,000 is negative

```
  111 111  11 1 11  1 1                          (carry)
0b01110111001101011001010000000000 +     (2,000,000,000)
0b01110111001101011001010000000000       (2,000,000,000)
==========
0b11101110011010110010100000000000        (-294,967,296)
```

Catching Overflow

- Why does the Java programming language ignore overflow?
  - Because most hardware doesn’t efficiently detect it and some can’t at all
  - The runtime would have to explicitly test for it on every add, subtract, and multiply
  - Use BigInteger if you might overflow
- C#, in contrast, has overflow detection as a debug runtime option

The Algorithms

XOR Swap

Credit Card Validation

Public Key Cryptography

Two’s Complement

Google MapReduce

Puzzle #5

- How can Google scale to such heights?

Answer

- With the help of MapReduce, a toolkit that simplifies the distribution of “parallelizable” work across hundreds or thousands of machines
- http://labs.google.com/papers/mapreduce.html

MapReduce

- The programmer specifies a “Map” rule and a “Reduce” rule
- Map: takes input key-value pairs and generates intermediate key-value pairs
- Reduce: consolidates intermediate pairs sharing the same key to a single set of values (usually one)
- Inspired by map and reduce in LISP

Distributed Grep

- Map
  - (line-number, line-string) → (line-number, line-string) or empty
  - Emit the number/string pair if it matches the pattern, otherwise ignore the pair
- Reduce
  - Identity function, just copy to output

Inverted Index

- Map
  - (document, words) → (word, document-id) as a series
  - From each document create a long list of word/document-id pairings
- Reduce
  - (word, list(document-id))
  - Gathers and sorts all refs to each word

Why?

- Why use MapReduce?
  - To operate beyond the CPU, memory, and disk limits of a single box
  - To abstract from the programmer the distribution logic
  - As well as fault handling, scheduling, monitoring
  - Let developers focus on the real problem

Implementations

- MapReduce was implemented by Google in C++ with Java programming language and Python language bindings
  - Runs on commodity Linux boxes with normal CPU and memory, local IDE disks
  - Google Filesystem (GFS) manages the data
- Apache Lucene has a Java programming language version called Hadoop
  - Uses the Hadoop Distributed FS (HDFS)

Execution Overview

- There's a master process to oversee a pool of workers
- Input gets split into chunks
- Chunks are assigned to workers, each worker performs the map logic on each pair found in the chunk
- Results are written locally and completed status is reported to the master

Execution Overview

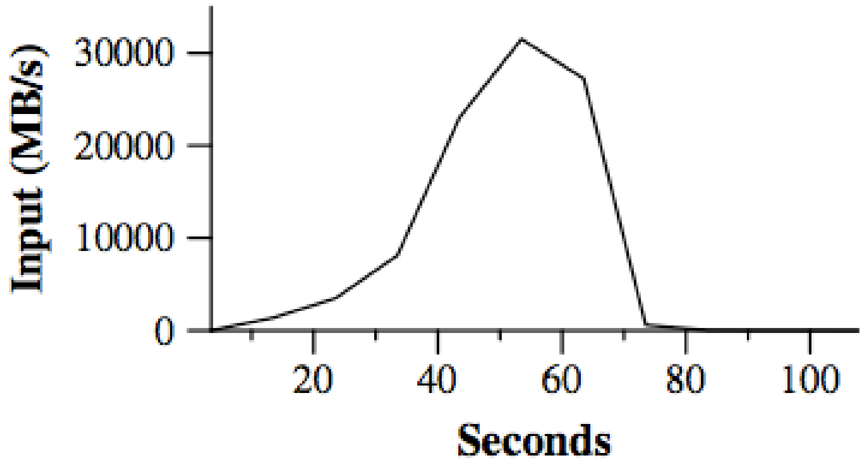
- The master assigns keys to reducers
- Partitioning dictates which reducer gets which key (i.e., a hash of the key)
- The reducer pulls the results using an iterator, sorts them, runs the reduce logic, and produces the “final” output
- Results are written to GFS
- Sometimes output goes through again

Distributed Grep Performance

- Scan 10^{10} 100-byte records searching for a rare 3-character pattern (92,337 hits)
  - Input split into 64 MB pieces, 15K chunks
  - Output all into a single file, one reducer

Distributed Grep

- Takes 150 seconds, with a peak input of 30G/sec when 1764 workers are assigned
- One minute of overhead, shuffling data



Redundant Execution

- A single slow worker, if it’s doing the last job (and it will be since it’s slow), can lengthen the completion time
- So near the end, spawn backup copies of tasks First worker to finish wins
- Testing shows times 30% speedup with this singular feature

Combiner Functions

- Programs like word-counting don’t need to send a pile of “1” values to the reducer
- Each map worker could reduce partially internally
- To do this you code a “combiner function”, usually the same as the reduce, but run on the map worker
- Huge speed up when semantics allow

Does Google Really Use MapReduce?

Number of jobs	29,423
Average job completion time	634 secs
Machine days used	79,186 days
Input data read	3,288 TB
Intermediate data produced	758 TB
Output data written	193 TB
Average worker machines per job	157
Average worker deaths per job	1.2
Average map tasks per job	3,351
Average reduce tasks per job	55
Unique *map* implementations	395
Unique *reduce* implementations	269
Unique *map/reduce* combinations	426

Table 1: MapReduce jobs run in August 2004

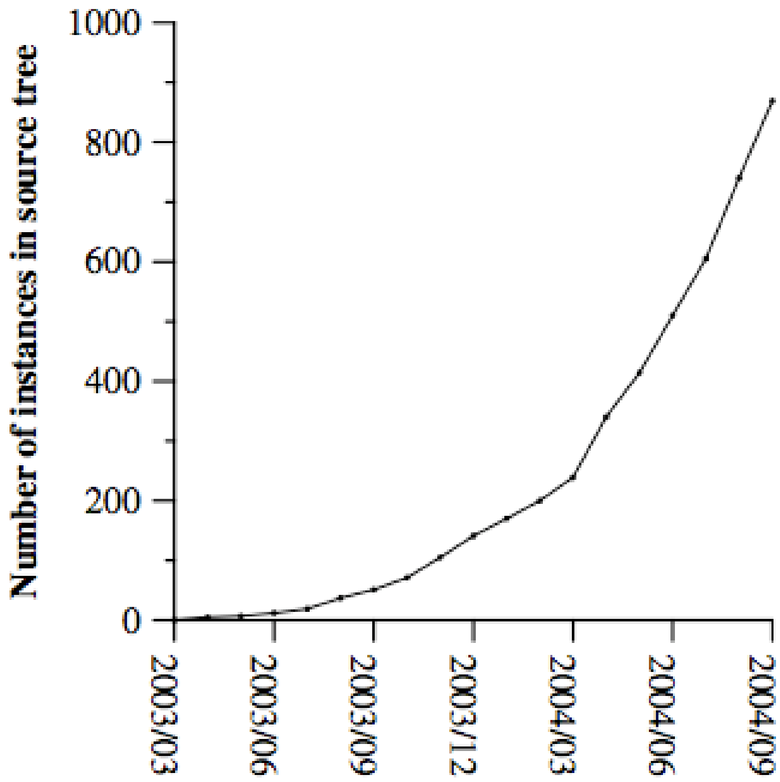


Figure 4: MapReduce instances over time

Links

- http://labs.google.com/papers/mapreduce.html
- http://lucene.apache.org/hadoop/about.html
- http://www.cs.vu.nl/~ralf/MapReduce/paper.pdf

Conclusion

- So now you can...
  - Swap values like a crazy person
  - Validate credit cards in the browser
  - Explain how RSA, 3DES, and SHA1 work
  - Work down deep in the bits of numbers
  - Or up high in massive parallel operations

Q&A

Jason Hunter

Web Algorithms

Jason Hunter

Principal Technlogist
Mark Logic
http://marklogic.com

Session TS-6045