CS162
Operating Systems and Systems Programming
Lecture 24

Security
Cloud Computing

April 29th, 2015

Prof. John Kubiatowicz

http://cs162.eecs.Berkeley.edu

What is Computer Security Today?

- Computing in the presence of an adversary!
  - Adversary is the security field's defining characteristic
- Reliability, robustness, and fault tolerance
  - Dealing with Mother Nature (random failures)
- Security
  - Dealing with actions of a knowledgeable attacker dedicated to causing harm
  - Surviving malice, and not just mischance
- Wherever there is an adversary, there is a computer security problem!

Protection vs. Security

- **Protection**: mechanisms for controlling access of programs, processes, or users to resources
  - Page table mechanism
  - Round-robin schedule
  - Data encryption
- **Security**: use of protection mech. to prevent misuse of resources
  - Misuse defined with respect to policy
    - » E.g.: prevent exposure of certain sensitive information
    - » E.g.: prevent unauthorized modification/deletion of data
  - Need to consider external environment the system operates in
    - » Most well-constructed system cannot protect information if user accidentally reveals password – social engineering challenge

Security Requirements

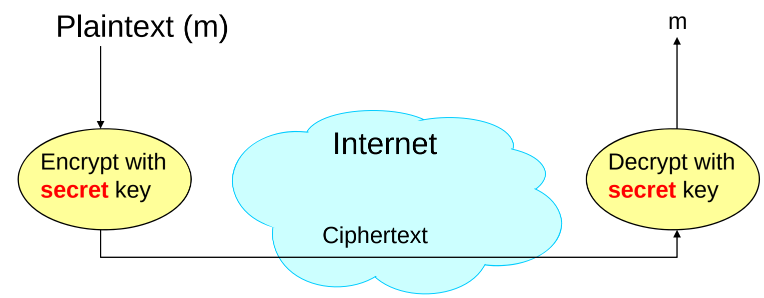
- Authentication
  - Ensures that a user is who is claiming to be
- Data integrity
  - Ensure that data is not changed from source to destination or after being written on a storage device
- Confidentiality
  - Ensures that data is read only by authorized users
- Non-repudiation
  - Sender/client can't later claim didn't send/write data
  - Receiver/server can't claim didn't receive/write data

Securing Communication: Cryptography

- Cryptography: communication in the presence of adversaries
- Studied for thousands of years
  - See the Simon Singh's The Code Book for an excellent, highly readable history
- Central goal: confidentiality
  - How to encode information so that an adversary can't extract it, but a friend can
- General premise: there is a key, possession of which allows decoding, but without which decoding is infeasible
  - Thus, key must be kept secret and not guessable

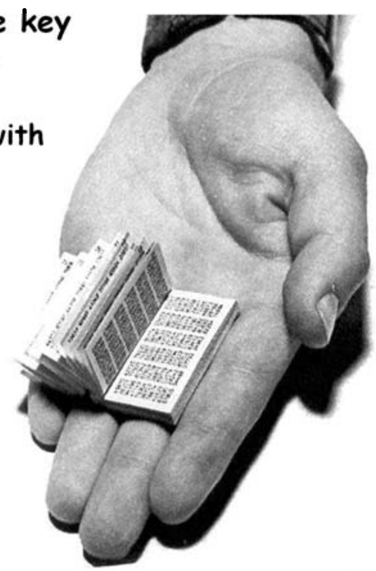
Using Symmetric Keys

- Same key for encryption and decryption
- Achieves confidentiality
- Vulnerable to tampering and replay attacks



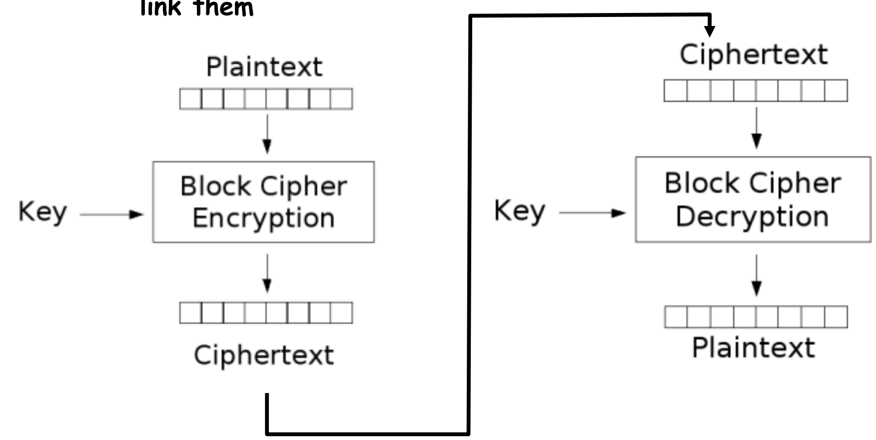
Symmetric Keys

- Can just XOR plaintext with the key
  - Easy to implement, but easy to break using frequency analysis
  - Unbreakable alternative: XOR with one-time pad
    - » Use a different key for each message



Symmetric Keys

- More sophisticated (e.g., block cipher) algorithms
  - Works with a block size (e.g., 64 bits)
    - » To encrypt a stream, can encrypt blocks separately, or link them

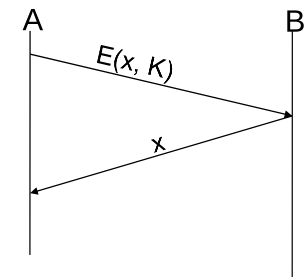


Symmetric Key Ciphers - DES & AES

- Data Encryption Standard (DES)
  - Developed by IBM in 1970s, standardized by NBS/NIST
  - 56-bit key (decreased from 64 bits at NSA's request)
  - Still fairly strong other than brute-forcing the key space
    - » But custom hardware can crack a key in < 24 hours
  - Today many financial institutions use Triple DES
    - » DES applied 3 times, with 3 keys totaling 168 bits
- Advanced Encryption Standard (AES)
  - Replacement for DES standardized in 2002
  - Key size: 128, 192 or 256 bits
- How fundamentally strong are they?
  - No one knows (no proofs exist)

Authentication via Secret Key

- Main idea: entity proves identity by decrypting a secret encrypted with its own key
  - K – secret key shared only by A and B
- A can asks B to authenticate itself by decrypting a nonce, i.e., random value, x
  - Avoid replay attacks (attacker impersonating client or server)
- Vulnerable to man-in-the middle attack

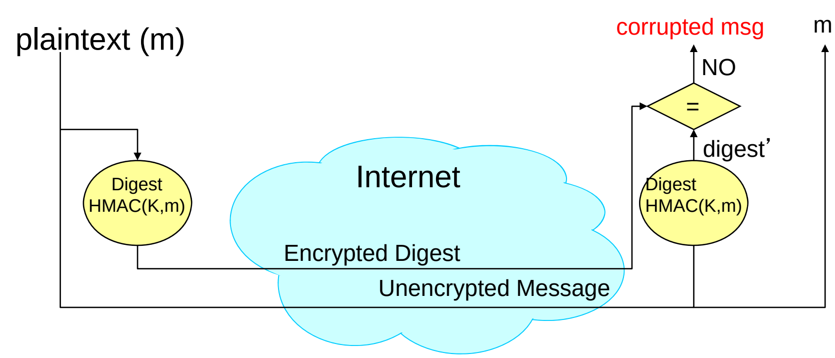


Notation: E(m,k) – encrypt message m with key k

Integrity: Cryptographic Hashes

- Basic building block for integrity: cryptographic hashing
  - Associate hash with byte-stream, receiver verifies match
    - » Assures data hasn't been modified, either accidentally – or maliciously
- Approach:
  - Sender computes a secure digest of message m using H(x)
    - » H(x) is a publicly known hash function
    - » Digest d = HMAC (K, m) = H (K | H (K | m))
    - » HMAC(K, m) is a hash-based message authentication function
  - Send digest d and message m to receiver
  - Upon receiving m and d, receiver uses shared secret key, K, to recompute HMAC(K, m) and see whether result agrees with d

Using Hashing for Integrity



Can encrypt m for confidentiality

Standard Cryptographic Hash Functions

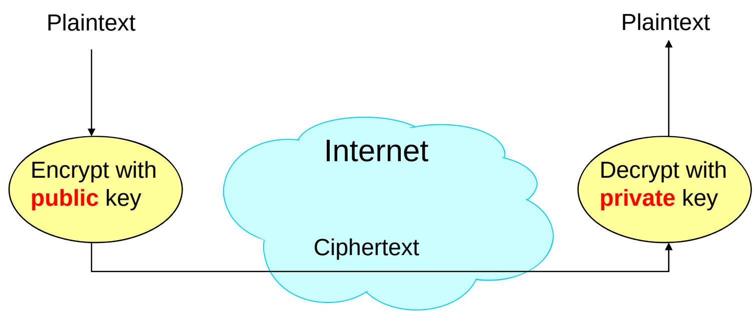
- MD5 (Message Digest version 5)
  - Developed in 1991 (Rivest), produces 128 bit hashes
  - Widely used (RFC 1321)
  - Broken (1996-2008): attacks that find collisions
- SHA-1 (Secure Hash Algorithm)
  - Developed in 1995 (NSA) as MD5 successor with 160 bit hashes
  - Widely used (SSL/TLS, SSH, PGP, IPSEC)
  - Broken in 2005, government use discontinued in 2010
- SHA-2 (2001)
  - Family of SHA-224, SHA-256, SHA-384, SHA-512 functions
- HMAC's are secure even with older "insecure" hash functions
  -

Asymmetric Encryption (Public Key)

- Idea: use two different keys, one to encrypt (e) and one to decrypt (d)
  - A key pair
- Crucial property: knowing e does not give away d
- Therefore e can be public: everyone knows it!
- If Alice wants to send to Bob, she fetches Bob's public key (say from Bob's home page) and encrypts with it
  - Alice can't decrypt what she's sending to Bob …
  - … but then, neither can anyone else (except Bob)

Public Key / Asymmetric Encryption

- Sender uses receiver's public key
  - Advertised to everyone
- Receiver uses complementary private key
  - Must be kept secret



Public Key Cryptography

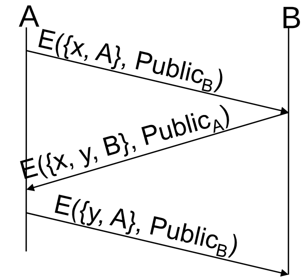
- Invented in the 1970s
  - Revolutionized cryptography
  - (Was actually invented earlier by British intelligence)
- How can we construct an encryption/decryption algorithm using a key pair with the public/private properties?
  - Answer: Number Theory
- Most fully developed approach: RSA
  - Rivest / Shamir / Adleman, 1977; RFC 3447
  - Based on modular multiplication of very large integers
  - Very widely used (e.g., ssh, SSL/TLS for https)
- Also mature approach: Eliptic Curve Cryptography (ECC)
  - Based on curves in a Galois-field space
  - Shorter keys and signatures than RSA

Properties of RSA

- Requires generating large, random prime numbers
  - Algorithms exist for quickly finding these (probabilistic!)
- Requires exponentiating very large numbers
  - Again, fairly fast algorithms exist
- Overall, much slower than symmetric key crypto
  - One general strategy: use public key crypto to exchange a (short) symmetric session key
    - » Use that key then with AES or such
- How difficult is recovering d, the private key?
  - Equivalent to finding prime factors of a large number
    - » Many have tried - believed to be very hard (= brute force only)
    - » (Though quantum computers could do so in polynomial time!)

Simple Public Key Authentication

- Each side need only to know the other side's public key
  - No secret key need be shared
- A encrypts a nonce (random num.) x
  - Avoid replay attacks, e.g., attacker impersonating client or server
- B proves it can recover x, generates second nonce y
- A can authenticate itself to B in the same way
- *Many more details to make this work securely in practice!*

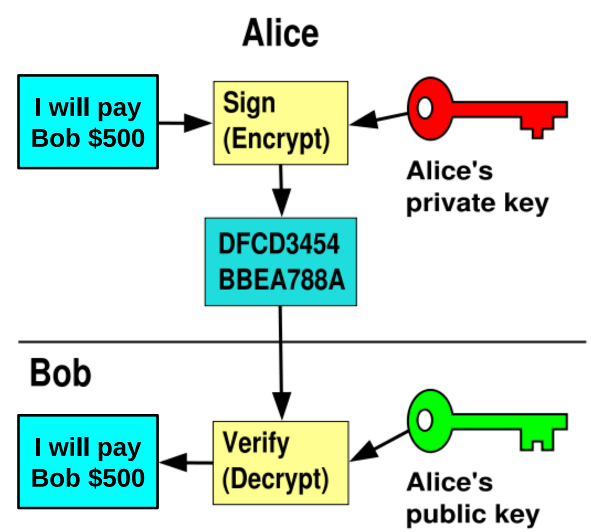


Notation: E(m,k) – encrypt message m with key k

Non-Repudiation: RSA Crypto & Signatures

- Suppose Alice has published public key KE
- If she wishes to prove who she is, she can send a message x encrypted with her private key KD (i.e., she sends E(x, KD))
  - Anyone knowing Alice's public key KE can recover x, verify that Alice must have sent the message
    - » It provides a signature
  - Alice can't deny it ⇒ non-repudiation
- Could simply encrypt a hash of the data to sign a document that you wanted to be in clear text
- Note that either of these signature techniques work perfectly well with any data (not just messages)
  - Could sign every datum in a database, for instance

RSA Crypto & Signatures (cont'd)



Digital Certificates

- How do you know K_E is Alice's public key?
- Trusted authority (e.g., Verisign) signs binding between Alice and K_E with its private key $KV_{private}$
  - $C = E(\{Alice, K_E\}, KV_{private})$
  - C: digital certificate
- Alice: distribute her digital certificate, C
- Anyone: use trusted authority's KV_{public}, to extract Alice's public key from C
  - $D(C, KV_{public}) = D(E(\{Alice, K_E\}, KV_{private}), KV_{public}) = \{Alice, K_E\}$

Summary of Our Crypto Toolkit

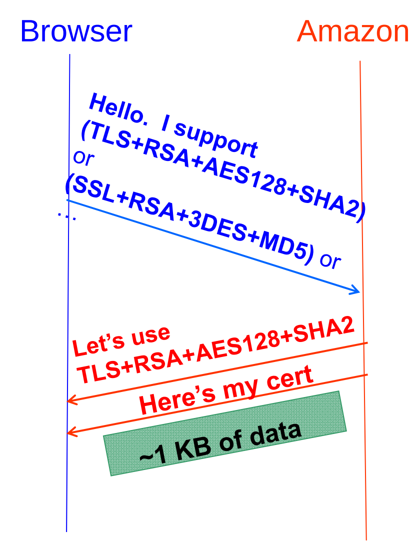
- If we can securely distribute a key, then
  - Symmetric ciphers (e.g., AES) offer fast, presumably strong confidentiality
- Public key cryptography does away with (potentially major) problem of secure key distribution
  - But: not as computationally efficient
    - » Often addressed by using public key crypto to exchange a session key
- Digital signature binds the public key to an entity

Putting It All Together - HTTPS

- What happens when you click on https://www.amazon.com?
- `https` = "Use HTTP over SSL/TLS"
  - SSL = Secure Socket Layer
  - TSL = Transport Layer Security
    - » Successor to SSL
  - Provides security layer (authentication, encryption) on top of TCP
    - » Fairly transparent to applications

HTTPS Connection (SSL/TLS) (cont'd)

- Browser (client) connects via TCP to Amazon's HTTPS server
- Client sends over list of crypto protocols it supports
- Server picks protocols to use for this session
- Server sends over its certificate
- (all of this is in the clear)



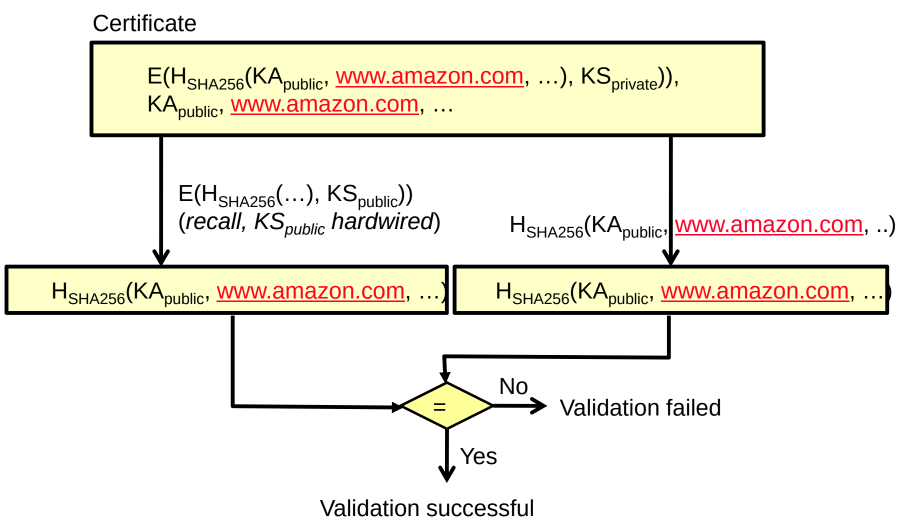
Inside the Server's Certificate

- Name associated with cert (e.g., Amazon)
- Amazon's RSA public key
- A bunch of auxiliary info (physical address, type of cert, expiration time)
- Name of certificate's signatory (who signed it)
- A public-key signature of a hash (SHA-256) of all this
  - Constructed using the signatory's private RSA key, i.e.,
  - Cert = $E(H_{SHA256}(KA_{public}$, www.amazon.com, …), $KS_{private}$))
    - » KA_{public}: Amazon's public key
    - » $KS_{private}$: signatory (certificate authority) private key
- …

Validating Amazon's Identity

- How does the browser authenticate certificate signatory?
  - Certificates of several certificate authorities (e.g., Verisign) are hardwired into the browser (or OS)
- If can't find cert, warn user that site has not been verified
  - And may ask whether to continue
  - Note, can still proceed, just without authentication
- Browser uses public key in signatory's cert to decrypt signature
  - Compares with its own SHA-256 hash of Amazon's cert
- Assuming signature matches, now have high confidence it's indeed Amazon …
  - … assuming signatory is trustworthy
  - *DigiNotar CA breach (July-Sept 2011): Google, Yahoo!, Mozilla, Tor project, Wordpress, … (531 total certificates)*

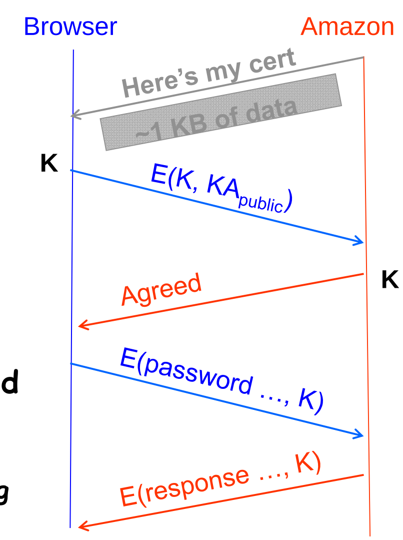
Certificate Validation



Can also validate using peer approach: https://www.eff.org/observatory

HTTPS Connection (SSL/TLS) cont'd

- Browser constructs a random ***session key*** K used for data communication
  - Private key for bulk crypto
- Browser encrypts K using Amazon's public key
- Browser sends $E(K, KA_{public})$ to server
- Browser displays
- *All subsequent comm. encrypted w/ symmetric cipher (e.g., AES128) using key K*
  - E.g., client can authenticate using a password



Administrivia

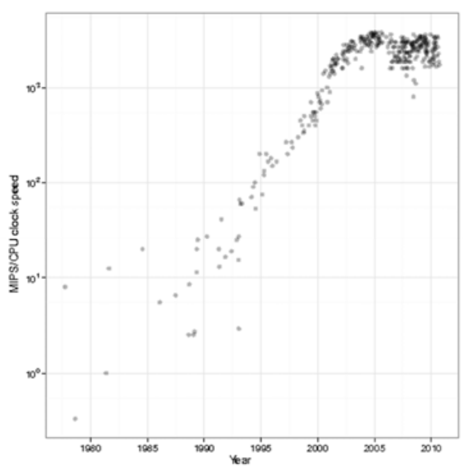
- Midterm 2 grading
  - In progress. To be done by Sunday
  - Solutions have been posted
- Project grades
  - Project 1 done by tomorrow
  - Project 2 done by middle of RRR
- Final Exam
  - Friday, May 15th, 2015.
  - 3-6P, Wheeler Auditorium
  - All material from the course
  - Two sheets of notes, both sides
  - Will need dumb calculator
- Targeted reviews: See posts on Piazza
  - Possibly 3 different sessions focused on parts of course

Administrivia (2)

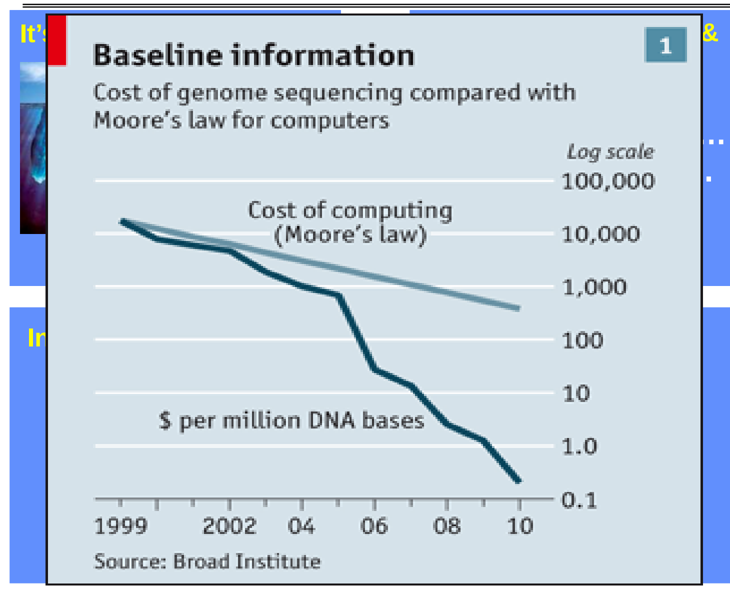
- Final topics (Monday, 5/4):
  - Go to poll on Piazza!
  - Current front runners:
    - » Internet of Things
    - » Quantum Computing
    - » Mobile OS

Background of Cloud Computing

- 1980's and 1990's: 52% growth in performance per year!
- 2002: The thermal wall
  - Speed (frequency) peaks, but transistors keep shrinking
- 2000's: Multicore revolution
  - 15-20 years later than predicted, we have hit the performance wall
- 2010's: Rise of Big Data

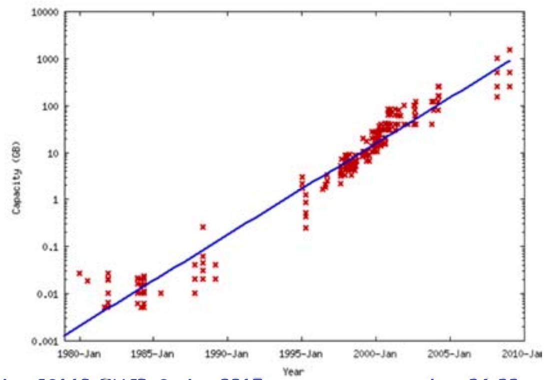


Sources Driving Big Data

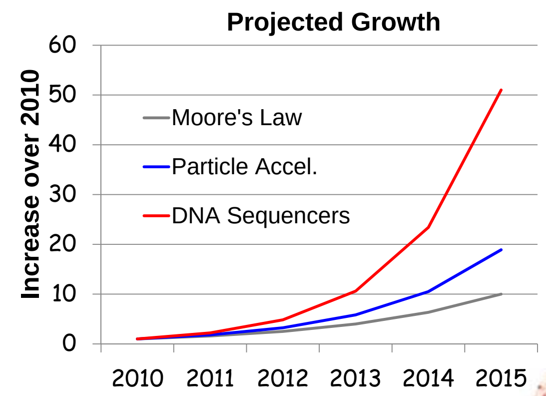


Data Deluge

- Billions of users connected through the net
  - WWW, FB, twitter, cell phones, …
  - 80% of the data on FB was produced last year
- Storage getting cheaper
  - Store more data!

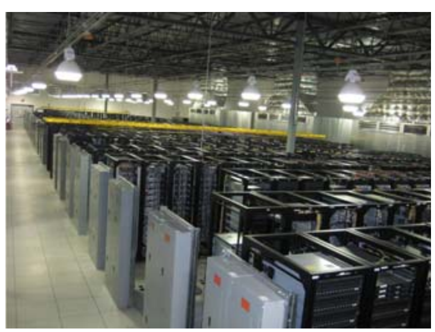


Data Grows Faster than Moore's Law



Solving the Impedance Mismatch

- Computers not getting faster, and we are drowning in data
  - How to resolve the dilemma?
- Solution adopted by web-scale companies
  - Go massively *distributed* and *parallel*

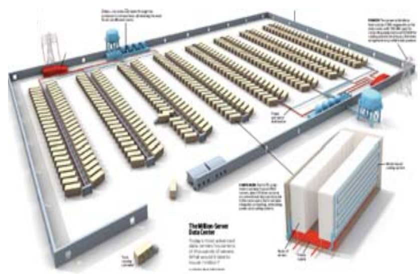


Enter the World of Distributed Systems

- Distributed Systems/Computing
  - *Loosely coupled* set of computers, communicating through message passing, solving a common goal
  - Tools: Msg passing, Distributed shared memory, RPC
- Distributed computing is *challenging*
  - Dealing with *partial failures* (examples?)
  - Dealing with *asynchrony* (examples?)
  - Dealing with *scale* (examples?)
  - Dealing with *consistency* (examples?)
- Distributed Computing versus Parallel Computing?
  - distributed computing ⇒
    parallel computing + partial failures

The Datacenter is the new Computer

- "The datacenter as a computer" still in its infancy
  - Special purpose clusters, e.g., Hadoop cluster
  - Built from less reliable components
  - Highly variable performance
  - Complex concepts are hard to program (low-level primitives)



= ?

Datacenter/Cloud Computing OS

- If the datacenter/cloud is the new computer
  - What is its Operating System?
  - Note that we are not talking about a host OS
- Could be equivalent in benefit as the LAMP stack was to the .com boom – every startup *secretly* implementing the same functionality!
- Open source stack for a Web 2.0 company:
  - Linux OS
  - Apache web server
  - MySQL, MariaDB or MongoDB DBMS
  - PHP, Perl, or Python languages for dynamic web pages

Classical Operating Systems

- Data sharing
  - Inter-Process Communication, RPC, files, pipes, …
- Programming Abstractions
  - Libraries (libc), system calls, …
- Multiplexing of resources
  - Scheduling, virtual memory, file allocation/protection, …

Datacenter/Cloud Operating System

- Data sharing
  - Google File System, key/value stores
  - Apache project: Hadoop Distributed File System
- Programming Abstractions
  - Google MapReduce
  - Apache projects: Hadoop, Pig, Hive, Spark
- Multiplexing of resources
  - Apache projects: Mesos, YARN (MapReduce v2), ZooKeeper, BookKeeper, …

Google Cloud Infrastructure

- Google File System (GFS), 2003
  - Distributed File System for entire cluster
  - Single namespace
- Google MapReduce (MR), 2004
  - Runs queries/jobs on data
  - Manages work distribution & fault-tolerance
  - Collocated with file system
- Apache open source versions:
  Hadoop DFS and Hadoop MR


The Google File System
Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google

MapReduce: Simplified Data Processing on Large Clusters
Jeffrey Dean and Sanjay Ghemawat
Google, Inc.

GFS/HDFS Insights

- *Petabyte* storage
  - Files split into large blocks (128 MB) and replicated across several nodes
  - Big blocks allow high throughput sequential reads/writes
- Data *striped* on hundreds/thousands of servers
  - Scan 100 TB on 1 node @ 50 MB/s = 24 days
  - Scan on 1000-node cluster = 35 minutes

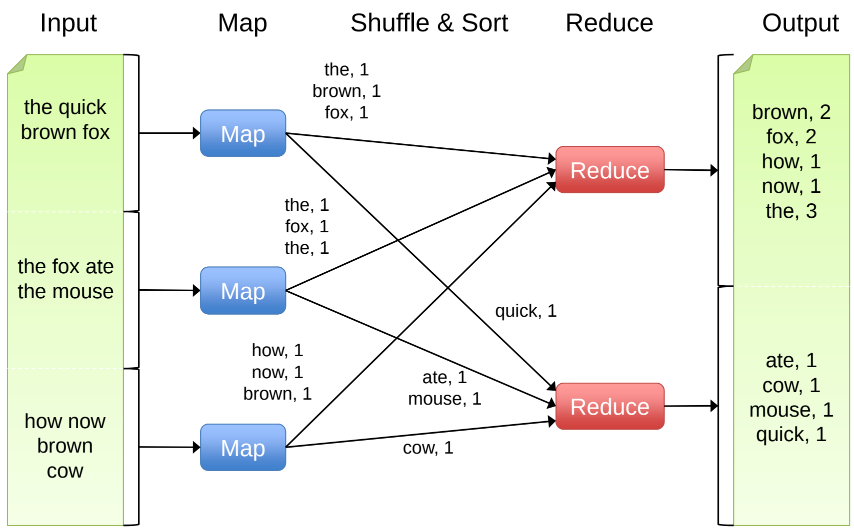
GFS/HDFS Insights (2)

- *Failures* will be the norm
  - Mean time between failures for 1 node = 3 years
  - Mean time between failures for 1000 nodes = 1 day
- Use *commodity* hardware
  - Failures are the norm anyway, buy cheaper hardware
- No complicated consistency models
  - Single writer, append-only data

MapReduce Programming Model

- Data type: key-value *records*
- Map function:
  $$(K_{in}, V_{in}) \rightarrow list(K_{inter}, V_{inter})$$
- Reduce function:
  $$(K_{inter}, list(V_{inter})) \rightarrow list(K_{out}, V_{out})$$

Word Count Execution



MapReduce Insights

- Restricted key-value model
  - Same fine-grained operation (Map & Reduce) repeated on big data
  - Operations must be deterministic
  - Operations must be idempotent/no side effects
  - Only communication is through the shuffle
  - Operation (Map & Reduce) output saved (on disk)

What is MapReduce Used For?

- At Google:
  - Index building for Google Search
  - Article clustering for Google News
  - Statistical machine translation
- At Yahoo!:
  - Index building for Yahoo! Search
  - Spam detection for Yahoo! Mail
- At Facebook:
  - Data mining
  - Ad optimization
  - Spam detection

MapReduce Pros

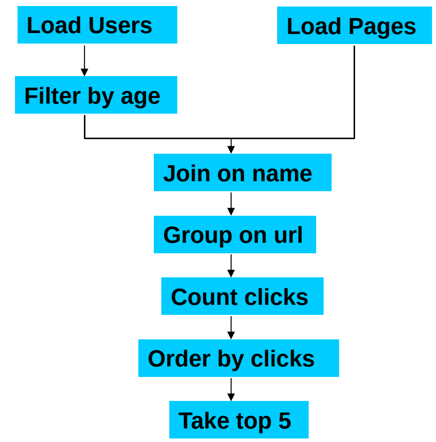
- Distribution is completely transparent
  - Not a single line of distributed programming (ease, correctness)
- Automatic fault-tolerance
  - Determinism enables running failed tasks somewhere else again
  - Saved intermediate data enables just re-running failed reducers
- Automatic scaling
  - As operations as side-effect free, they can be distributed to any number of machines dynamically
- Automatic load-balancing
  - Move tasks and speculatively execute duplicate copies of slow tasks (*stragglers*)

MapReduce Cons

- Restricted programming model
  - Not always natural to express problems in this model
  - Low-level coding necessary
  - Little support for iterative jobs (lots of disk access)
  - High-latency (batch processing)
- Addressed by follow-up research and Apache projects
  - Pig and Hive for high-level coding
  - Spark for iterative and low-latency jobs

Example Problem

Given *user data* in one file, and *website data* in another, find the *top 5 most visited pages by users aged 18-25*



Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

In MapReduce

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

Apache Pig

- High-level language:
  - Expresses sequences of MapReduce jobs
  - Provides relational (SQL) operators (JOIN, GROUP BY, etc)
  - Easy to plug in Java functions
- Started at Yahoo! Research
  - Runs about 50% of Yahoo!'s jobs
- https://pig.apache.org/
- Similar to Google's (internal) Sawzall project

In Pig Latin

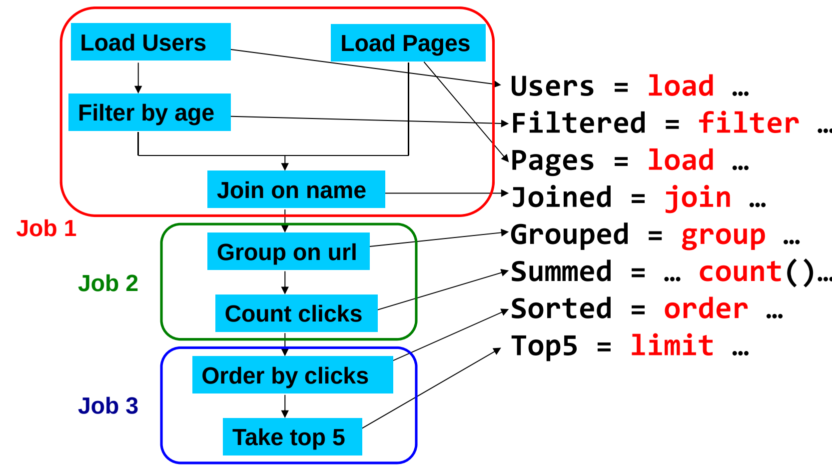
```
Users    = load 'users' as (name, age);
Filtered = filter Users by
                  age >= 18 and age <= 25;
Pages    = load 'pages' as (user, url);
Joined   = join Filtered by name, Pages by user;
Grouped  = group Joined by url;
Summed   = foreach Grouped generate group,
                   count(Joined) as clicks;
Sorted   = order Summed by clicks desc;
Top5     = limit Sorted 5;

store Top5 into 'top5sites';
```

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

Translation to MapReduce

Notice how naturally the components of the job translate into Pig Latin

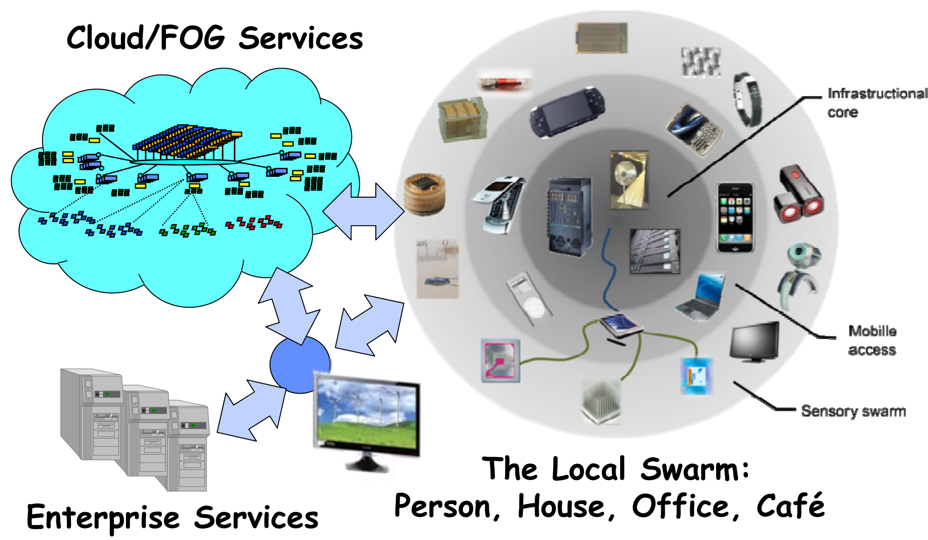


Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

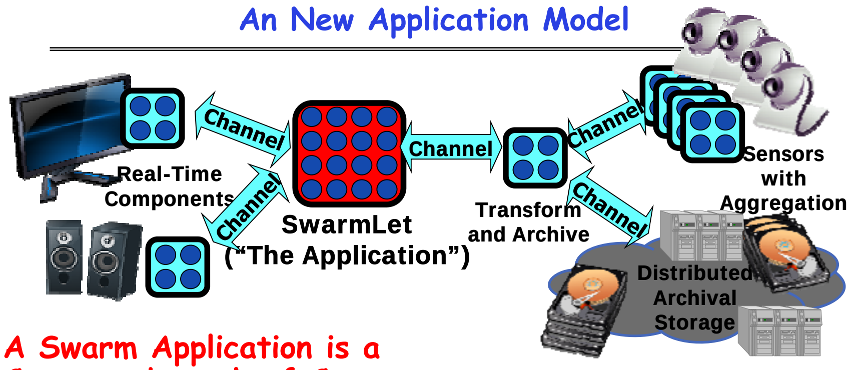
Future?

- Complete location transparency
  - Mobile Data, encrypted all the time
  - Computation anywhere any time
  - Cryptographic-based identities
  - Large Cloud-centers, Fog Computing
- Internet of Things?
  - Everything connected, all the time!
  - Huge Potential
  - Very Exciting and Scary at same time
- Better programming models need to be developed!
- Perhaps talk about this on Monday

Truly Distributed Apps: The Swarm of Resources

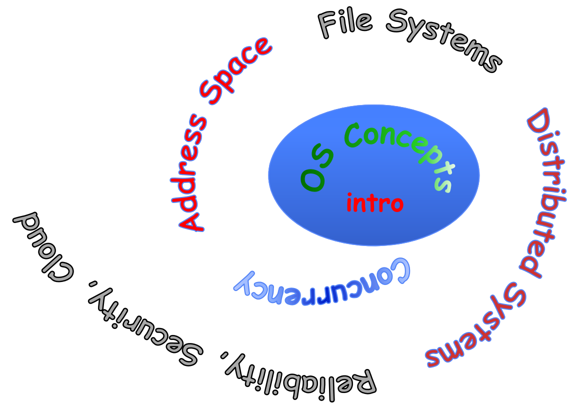


An New Application Model



- A Swarm Application is a Connected graph of Components
  - Globally distributed, but locality and QoS aware
  - Avoid Stovepipe solutions through reusability
- Many components are *Shared Services* written by programmers with a variety of skill-sets and motivations
  - Service Level Agreements (SLA) with micropayments

Thank you!



- Let's Thank the TAs!
- Thanks for helping us with this experimental version of the course… I think that it is going to be great!
- Good Bye!