# Distributed Systems — Introduction

Allan Clark

School of Informatics
University of Edinburgh

http://www.inf.ed.ac.uk/teaching/courses/ds
Autumn Term 2012

# Distributed Systems — Definitions

- *"A system in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing." — Coulouris*
- *"A system that consists of a collection of two or more independent computers which coordinate their processing through the exchange of synchronous or asynchronous message passing."*
- *"A distributed system is a collection of independent computers that appear to the users of the system as a single computer." — Tanenbaum*
- *" A distributed system is a collection of autonomous computers linked by a network with software designed to produce an integrated computing facility."*

## Computer Networks vs. Distributed Systems

- Computer Network: the autonomous computers are explicitly visible — have to be explicitly addressed
- Distributed System: existence of multiple autonomous computers is transparent
- The study of computer networks is concerned with how to send messages between machines, whilst the study of distributed systems is how to use those networks to get stuff done.
- However,
  - many problems in common,
  - in some sense networks (or parts of them, e.g., name services) are also distributed systems, and
  - normally, every distributed system relies on services provided by a computer network.

# Reasons for Distributed Systems

- Inherent distribution stemming from the application domain, e.g.
  - cash register and inventory systems for chain-stores
  - computer supported collaborative work
  - multi-player games
- Resource sharing is often a strong motivation
- Load distribution
  - amazon.com is not a single computer
  - these separate computers can be turned on-off for different demand profiles
- Critical failure tolerance, e.g. peer-to-peer networks
  - amazon.com isn't even located on a single site.
  - It is therefore resilient to (to some extent) earthquakes, power outages and more mailicious attacks

# Consequences

These may be good, bad or somewhere in between:

- Software - how to design and manage it in a distributed system
- Dependency on the underlying network infrastructure
- Easy access to shared data raises security concerns
- Emergent behaviour, sometimes good, bad, or just fascinating

# Consequences

- Distributed systems are concurrent systems
  - This concept will come up again and again
  - Synchronization and coordination by message passing
  - Sharing of resources, as both a positive and a negative
  - Typical problems of concurrent systems
    - Deadlocks and Livelocks
    - Unreliable communication
- Absence of a global clock
  - Due to asynchronous message passing there are limits on the precision with which processes in a distributed system can synchronize their clocks

# Consequences — continued

- Absence of a global state
  - In the general case, there is no single process in the distributed system that would have a knowledge of the current global state of the system
    - Due to concurrency and message passing communication
- Specific failure modes
  - Processes run autonomously, in isolation
    - Failures of individual processes may remain undetected
    - Individual processes may be unaware of failures in the system context

# Emerged/emerging Distributed Systems

1. Commerce
2. Encyclopedias (more generally knowledge stores)
3. Publishing in general
4. Finance
5. Education
6. Science
7. Healthcare

# Examples of Distributed Systems

## Web Search

- Google's infrastructure is one of the world's largest installations of a distributed system. It must visit and index a ridiculously large volume of web content in a variety of formats and then index this content for speedy results.
- Any numbers I give would be out of date tomorrow and are in any case unimaginable
- 68 Billion pages, maybe
- Data centres around the world
- A distributed file system designed for very fast access to very large files

# Examples of Distributed Systems

## (Massively) Multiplayer Games

- A particular need for fast response times
- Propogation of events and maintenance of the universe (or global state).
- The consequences of failure are potentially not as bad for the users (though major loss of revenue for the vendors)
- Most commericial offerings depend upon large infrastructure whether that be centrally managed or more distributed
- But, we are seeing the emergence of peer-to-peer based architectures for online games, with each user contributing some resources
- As such online games can be seen as a testbed for distributed systems (as they have proven in the past)

# Examples of Distributed Systems

## Online Betting

- Clearly betting has moved from the high street to the Internet
- More importantly there are now examples of distributed "layers" or "bookmakers"
- Examples are `betfair.com` and `intrade.com`
- Traditionally a bookmaker (using a greybeard and mathematics) would "set" or "fix" the odds for each particular bet
- Distributed bookmakers allow anyone to "back" or "lay" any particular bet (or market) at any particular price
- For example I can offer odds that Stoke City will win the EPL this year at odds of 1 in 4
- Sadly it is unlikely that anyone will take up this offer.
- Odds emerge as a market outcome

# Examples of Distributed Systems

## Financial Markets

- On the forefront of distributed systems development
- Due to a need for real-time information from a multitude of sources
- Have a need to relay events to potentially large numbers of clients
- For this reason they have unsual underlying architectures
- Emergent behaviour can be undesirable here, e.g. flash crash 2:45

# Examples of Distributed Systems

## Financial Markets

- On the forefront of distributed systems development
- Due to a need for real-time information from a multitude of sources
- Have a need to relay events to potentially large numbers of clients
- For this reason they have unsual underlying architectures
- Emergent behaviour can be undesirable here, e.g. flash crash 2:45
  - Thursday 6th May 2010
  - Dow Jones industrial average plunged approximately 1000 points
  - This was about 9% at the time
  - The largest one day point decline ever
  - The losses were recovered within minutes
  - Nobody knows to this day what happened

# Examples of Distributed Systems

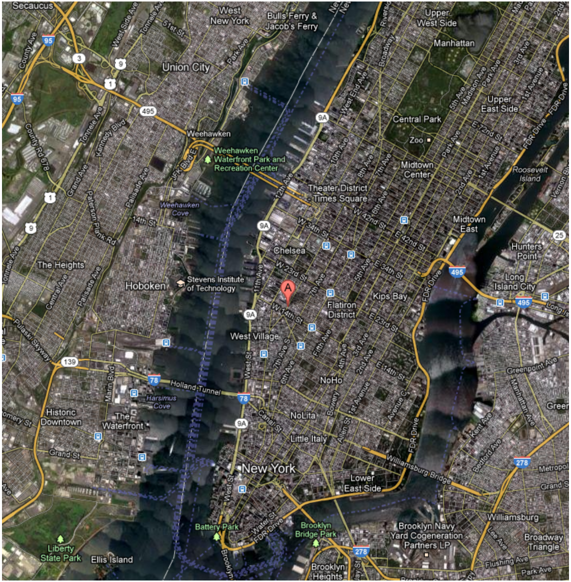
# Examples of Distributed Systems


Google bought this place for 1.9 billion dollars

# Examples of Distributed Systems

| Building | Location | Height | Built | Price (USD) |
|---|---|---|---|---|
| The Shard | London, UK | 310 metres | 2012 | 3.9b |
| Antilla | Mumbai, India | 173 metres | 2007/10 | 2b |
| Taipei 101 | Taipei, Taiwan | 509 metres | 2004 | 1.76b |

# Examples of Distributed Systems



# Examples of Distributed Systems

## Source Code Control

- Source code control is the endeavour to maintain a full history of changes to a project's source code, often by multiple authors
- Only the original source code and the changes (or diffs) are stored
- Concurrent updates are allowed when different parts of the code are changed, in which case the changes can be "merged"
- Where the same part is changed concurrently there is a "conflict" which must be resolved before operations may continue
- This allows for multiple versions of the source code such as a release and development branch
- Bugs can be tracked down to the change in which the bug was introduced, thereby elminating many possible causes

# Examples of Distributed Systems

## Source Code Control

- This has always tended to be a distributed system
- In the sense that there are multiple authors
- Traditionally there was a client-server based architecture
- One centralised server with the single repository
- Authors request:
  - New revisions
  - That their revisions be recorded in the global history

# Examples of Distributed Systems

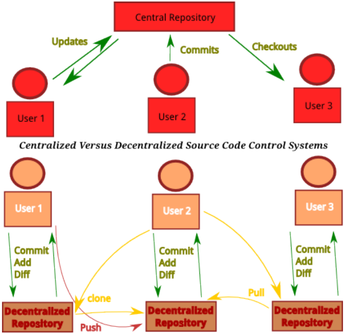
## Source Code Control

- Recently (last decade or so) source code control systems have been decentralised or distributed
- Each contributor clones the entire history and has a local repository.
- Revisions can be sent and received between any two repositories
- There is greater fault tolerance, if the original centralised server fails a different one is simply declared the new master
- Merging can occur between smaller groups before commiting to a larger audience (of the master repository)

# Examples of Distributed Systems

## Source Code Control

- Centralised: cvs, subversion, ClearCase, Vault
- Distributed: git, mercurial, darcs, bazaar, bitkeeper



# Challenges in Design of Distributed Systems

- 1. Heterogeneity
  - Hardware, Networks, Operating Systems, Programming Languages
  - Not just heterogeneity of implementation but sometimes of characteristics such as reliability or speed.
  - In a sense this much of this is a networking problem, that is the difficulty of sending messages around heterogeneous networks
  - But it does have implications, such as I have mentioned before for software versioning
  - Approaches generally use abstraction
    - Middleware (e.g., CORBA): transparency of network, hard- and software and programming language heterogeneity
    - Mobile Code (e.g., JAVA): transparency from hard-, software and programming language heterogeneity through virtual machine concept

# Challenges in Design of Distributed Systems

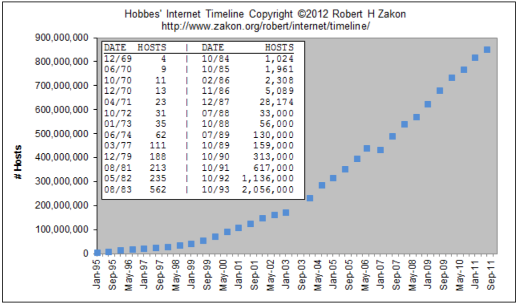
- 2. Openness
  - How open a distributed system is determines whether it can be extended domain, both size and functionality
  - Mostly determined by how well published are the interfaces which are used
  - Many web-services are being turned into mobile applications because they have well defined and published interfaces
  - An open system is less reliant on a particular vendor
- 3. Security,
  - has essentially three main components:
    1. Confidentiality — protection against access by unauthorised individuals
    2. Integrity — protection against alteration or corruption
    3. Availability — protection against loss of access whether circumstantial or a malicious denial of service attack
  - Security forms a later part of this course — but in summary, encryption only gets you part of the way there

# Challenges in Design of Distributed Systems

- 4. Scalability
  - Does the system remain effective given expectable growth?
  - Expectable growth of physical resources <u>and</u>
  - Expectable growth of users
  - Avoiding Performance bottlenecks
    - Early Domain Name Lookup consisted of a single centrally hosted file
    - The "hosts.txt" file mapped names to numerical addresses
    - Client computers were required to periodically re-download this file from its known location (at SRI, now SRI International)
    - The "hosts.txt" file still exists on most operating systems today and can be used for much hilarity if you can access your friend's hosts.txt file
    - $ dig www.some-annoying-site.com ⇒ 173.194.67.103
    - 173.194.67.103 www.bbc.co.uk

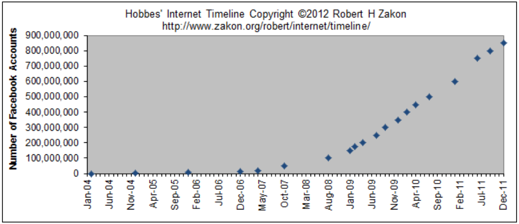
# Challenges in Design of Distributed Systems

- 4. Scalability
  - After DNS was developed:
  - Some time in the late 1970s it was decided that 32 bit addresses would be enough, but they are currently running out.
  - IP addresses are in the process of switching from 32 bit addressing to 128 bit addressing but overcompensating could have been a serious performance issue



# Challenges in Design of Distributed Systems

- 4. Scalability
  - Expectable growth is often non-obvious



# Challenges in Design of Distributed Systems

- 5. Handling of failures
  - Detection (may be impossible)
  - Masking
    - retransmission
    - redundancy of data storage
    - generally not guaranteed in the worst case
  - Tolerance
    - exception handling (e.g., timeouts when waiting for a web resource)
  - Recovery
    - Can be especially tough, the failed process may have left some permanent data in an inconsistent state
  - Redundancy
    - redundant routes in network
    - replication of name tables in multiple domain name servers

# Challenges in Design of Distributed Systems

- 6. Concurrency
  - Consistent scheduling of concurrent threads (so that dependencies are preserved, e.g., in concurrent transactions)
  - Avoidance of dead- and livelock problems
  - Actions are concurrent if A may happen before B and B may happen before A
  - Generally you hope for consistent results in either case
  - I will have more to speak about concurrency

# Challenges in Design of Distributed Systems

- 7. Transparency: concealing the heterogeneous and distributed nature of the system so that it appears to the user like one system.
  - Transparency categories (according to ISO's Reference Model for ODP)
    - Access: access local and remote resources using identical operations e.g., network mapped drive
    - Location: access without knowledge of location of a resource e.g., URLs, email addresses
    - Concurrency: allow several processes to operate concurrently using shared resources in a consistent fashion
    - Replication: use replicated resource as if there was just one instance
    - Failure: allow programs to complete their task despite failures e.g., retransmit of email messages
    - Mobility: allow resources to move around
    - Performance: adaption of the system to varying load situations without the user noticing it
    - Scaling: allow system and applications to expand without need to change structure or application algorithms

# Any Questions

Any Questions?