# Unit 1 - Introduction

Network Information System Technologies

# Goals

- Networked systems are Distributed Systems
- Get an understanding of
  - What is a distributed system
  - Why are they relevant
  - What are the main applications
- Additionally
  - Explore some examples
  - Get a good grasp on the important topic of Cloud Computing

# Index

# 1. What is a Distributed System

- Set of autonomous agents
  - Each agent is a sequential process, proceeding at its own pace.
- Agents interact. Options:
  - Message passing → Better
  - Shared memory → RACE CONDITIONS !!!
- Agents have their own independent state
- There is some collective goal to this cooperation
  - By which the behaviour of the "system" can be assessed.
- In practice, a Distributed System is a Networked System.

# Index

# 2. Relevance

- Evolving field since its beginnings
  - Offshoot of concurrent systems
    - Heavily studied for their usefulness in the design of time sharing systems
    - You should be familiar with many aspects of concurrent systems (CSD)
- Pushed by evolution of computer networks
    - How to make all those computers do something globally useful?

Every distributed sys. are Concurrent

BUT

There are concurrent sys. that are nt distributed

# 2. Relevance

- Main "reasons" (as presented back in the 80s)

THROUGHPUT INCREASES | For READ accesses → throughput increases proportionally to the nº of servers

1. Speed up → Depends on operation → | For UPDATES → Throughput cannot be increased linearly with additional cases
   - Take a complex problem, split it in pieces, have each piece taken care of by a different computer.
2. Fault tolerance. Basic idea:
   - If one computer breaks down, we still have other computers capable of carrying out the tasks of the crashed one.
3. Resource Sharing
   - One computer may have resources (e.g., printers, disks, …) other computers do not have (and do not need to have)
   - It should be possible to access resources from everywhere

ALSBERG & DAY → Passive → Sequential consistency

ALSBERG & DAY → Multi-master → Eventual consistency

# 2. Relevance

- All the previous reasons are still valid today because the computing environment IS distributed and interconnected
  - Myriad of connected "computers"
  - Myriad of remote services
    - Accessed as shared resources
    - Everyone knows and uses the WWW
- Challenges
  - Leverage the connectivity to achieve useful results
  - Create subsystems capable of delivering well-behaved services
    - How does Google manage to deliver their search engine?
    - How does Dropbox manage to serve millions of users shared files
    - … (your favourite service goes here)

# Index

1. Concept of Distributed System
2. Relevance
3. Application Areas
4. Back to the beginning: Cloud Computing
5. Programming Paradigms
6. Conclusions: No computing without a Network

# 3. Application Areas

- Some important DISTRIBUTED application areas are:
  1. *World Wide Web*
  2. Sensor networks
  3. *Internet of Things*
  4. Cooperative computing
  5. Highly available *Clusters*
- They are described on the sequel…

# 3.1. Application Areas: The WWW

- Based on the Client-Server Model
- Server attends requests for documents
  - Requests may involve reading or writing of a document
- The Clients are Browsers, sending/receiving documents
  - Browsers parse documents searching for metadata
  - Links are particular metadata pointing to other documents
    - Documents may be in another server
- Simple and powerful paradigm
  - Initially conceived for <u>document sharing</u>
  - Extended to allow document requests to stand for general service requests
    - Returned "documents" encode the result of the actual request

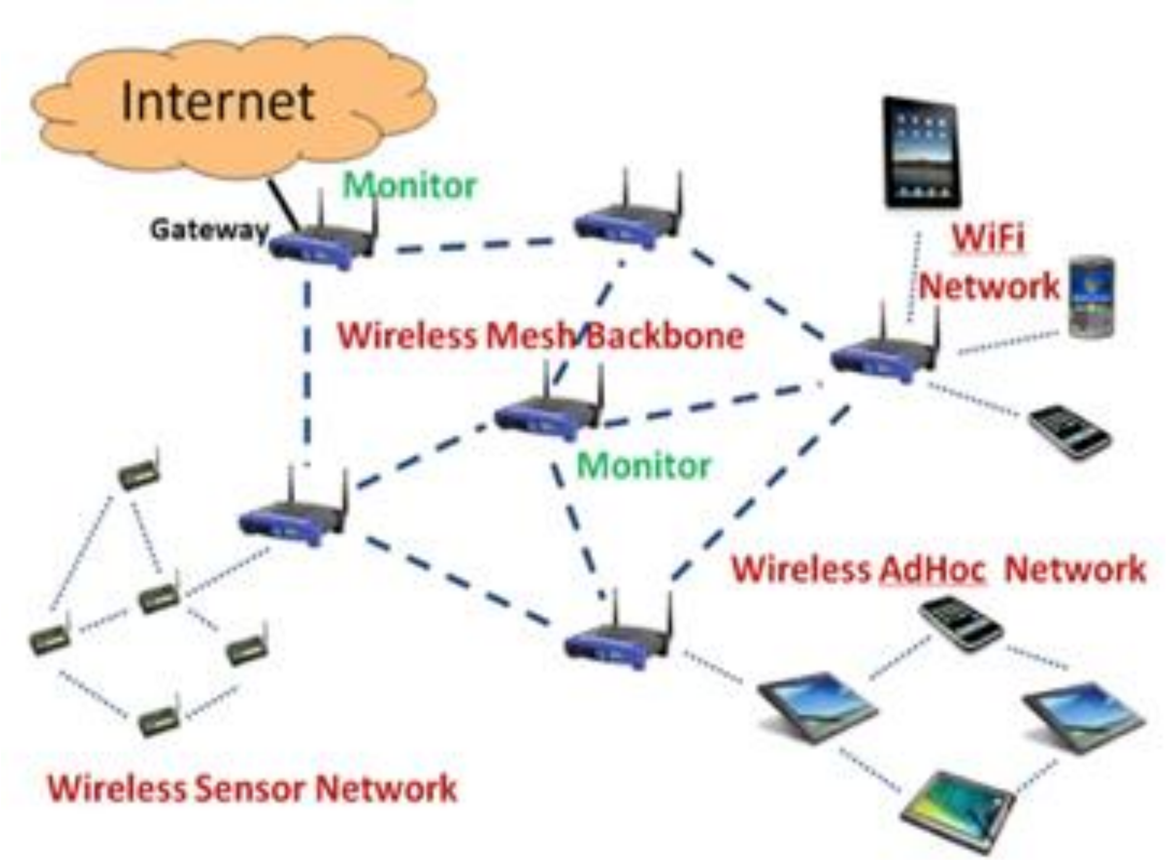
## 3. 2. Application Areas: Sensor Networks

- Driven by declining costs of hardware
- Special purpose mini-computers
  - Motes
- Embedded in common devices
  - Dishwashers, etc
- Contain physical world sensors
  - Humidity, temperature, power consumption, …
- Wide range of potential applications
  - Surveillance
  - Biological and chemical disaster detection
  - Power monitoring
  - …

# 3. 3. Application Areas: The "Internet of Things"

- Motivation: leverage ubiquitous connectivity of all devices
  - Generalization of sensor networks
    - All devices can, and will interact among them
    - Devices can also alter their physical environment
  - New scenarios open up
    - Smart cities
    - Building automation
    - Healthcare
    - …

# 3.3. Application Areas: The "Internet of Things"



## 3.4. Application Areas: Cooperative Computing

- Most computational power is underused
  - The desktops spend many hours doing nothing
- Many engineering and scientific problems can be split into pieces (tasks)
  - Each task can be resolved in a small amount of time
  - Results from each piece can be composed to complete the resolution of the whole problem
- Servers can be set up with an instance of such a problem
  - The server creates a pool of smaller tasks
- Computers across the internet can subscribe to receive tasks to solve
  - They install a special client software: the task runtime environment
  - The client registers with the server
- The server spreads tasks among the registered clients, and collects their results

# 3. 5. Application Areas: Highly Available Clusters

- So far we have seen application areas addressing resource sharing and cooperation.
- Fact:
  - Devices fail. Computers are devices. They fail at some point with a 100% probability.
- Fact:
  - Not all devices fail at the same time, always.
  - Q: when can it happen?
- Some environments need a high degree of availability
  - Banking
  - Finances
  - …
- Leverage having more than one device to stand failures

# 3.5. Application Areas: Highly Available Clusters

- HA Cluster:
  - Set of computers, with server programs on which clients depend constantly
  - Typically holding sensitive data
  - Designed with specific protocols to stand failures of one or more of them
  - Two main concerns:
    - Preserve data integrity
    - Preserve server operation availability

# 3.6. Application Areas: Cloud Computing

- Main current trend to build and provide services
- Fact:
  - Computer power in the traditional usage is underused
    - We already discussed this earlier
  - Setting up enterprise-class computer centres, with their applications is expensive:
    - Purchasing the software and hardware
    - Paying up the engineers which manage hardware and software
    - Paying for energy consumption
      - Onerous, when resources are underutilized

… Let us dive into CC…

# Index

1. Concept of Distributed System
2. Relevance
3. Application Areas
4. Back to the beginning: Cloud Computing
5. Programming Paradigms
6. Conclusions: No computing without a Network

# 4. CC: Cloud Computing

- Items to be discussed:

1. Programs and services
2. Roles in the service life cycle
3. Evolution of software services
   a) Mainframes
   b) Personal computers
   c) Enterprise computer centres
   d) SaaS
   e) IaaS
   f) SaaS on IaaS
   g) PaaS
4. Summary

# 4.1. Software and Services

- Generic Goal of CC:
  - Make creation and exploitation of services based on software simpler and more efficient
- An obvious fact:
  - Software has always been made to get some sort of service
  - Helped by hardware, of course
- Computer industry evolution helped to obscure this fact:
  - Personal Computer industry has imposed a specific mode of interaction of users with their computers.

# 4.2. Roles in the Service Life Cycle

- Consider the following four roles:
  - Developer
    - Gets the software components built
  - Service provider
    - Decides the characteristics of a service, the components that make it up, and how it should be configured and managed
  - System's administrator
    - Makes sure every piece of software/hardware is in place and properly configured
  - Service user
    - Accesses the service

# 4.3. Service Evolution: a) Mainframes

- *System administration* taken care of by specialists
- Very few tensions of contention
  - Small enough user base
- Efficient use of hardware
  - Shared by a large population of users
  - Low up-front cost for a user
    - Purchase cost born by Mainframe owner
- Mixed role for Users
  - Many were also the developers
  - Many were also their own service providers
    - With the software they developed
    - With third party software
- Users were involved in too many of the management details for the services they finally used

# 4.3. Service Evolution: b) Personal Computers/Workstations

- Trends with increased computer power produced the personal computer
  - Users no longer needed to have access to a mainframe in a computer centre
- No contention
  - Precisely one of the selling points of the model
- Wasteful use of resources: computer infra-utilized
- Up-front investment cost to purchase
- Rationalization of the role of developer
  - Specialized organizations build the software
- But still, mixed role of user
  - Works as service provider
    - Selects the mix of software she needs to get some job done
  - Works as system administrator of her PC
- Too much complexity for the majority of users

# 4.3. Service Evolution: c) Enterprise Computer Centers

- Including HA Cluster set-ups
- Similar characteristics to the PC situation
  - The user being the enterprise
  - High Personnel cost to take up the roles of system administrator and service provider
  - On occasion, developer role, for in-house software
- Variation based on hosting the software on external Data Centre
  - Avoid up front hardware costs
  - Avoid costs of hardware maintenance and management
  - Avoid constant power usage costs
  - Better able to manage computing costs

# 4.3. Service Evolution:
# d) Software as a Service (SaaS)

- Service is accessed through the network.
  - Typically using a web browser
- Clear separation of the role of user
  - Service is defined by a third party: the service provider
- Not so clear separation of the other roles
  - Software is initially developed mostly by the provider
  - All management aspects are taken care of by the provider
    - Including management of hardware in data centres
    - Including management of the installed software on the hardware
- Initially some inefficiencies
  - Lack of flexibility in hardware allocation
    - Leads to provider committing to a certain resource usage.
    - Diminishes resource sharing
  - Limited contention: resource reservation for expected demand

# 4.3. Service Evolution: d) SaaS

- Driven by
  - Improvement of networking technology
    - Higher bandwidth
    - Lower latency
  - Built capacity of Data Centres
    - Made it attractive to host services for external users
  - Technology improvements in the Browser
    - Epitomized by the buzzword "Web 2.0"
    - Browser capable of executing locally complex user interactions, delivered via Browser-executing software
      - Allowed attractive user interfaces
      - Improved scalability:
        - Workload reduction on servers.

# 4.3. Service Evolution: e) Infrastructure as a Service (IaaS)

- Provides ability to easily allocate/de-allocate computer and network resources on demand
  - Requests via API to a service (the IaaS service)
  - Ability to load custom OS images on those computers
  - Ability to request concrete computer and network capacities
- Driven by Hardware Virtualization technology
  - Easy and fast allocation/deallocation of hardware resources (virtual)
  - Easy sizing of hardware resources (virtual)
  - Easy set-up of a computer image within a virtual machine

# 4.3. Service Evolution: f) SaaS on IaaS

- IaaS introduces pay-as-you-go model
  - A core characteristic of Cloud Computing
- Makes it feasible to create SaaS which adapt to their user's load
  - The more user load, the more hardware resources it requests
    - *Elasticity:* another Cloud characteristic
  - Moves the pay-as-you-go model to the SaaS
    - Users of SaaS also pay per use of the service
- Very efficient usage of resources for SaaS providers
  - Most costs are variable
  - No up-front costs to reserve capacity (purchase or commitments)
  - Savings passed to SaaS users: competitive market of services.

# 4.3. Service Evolution: f) SaaS on IaaS

- IaaS providers take the risk of up-front investment
  - Promise of a large population of SaaS providers
    - Themselves with large populations of SaaS users
    - Large demand of virtualized resources
- SaaS provider still has mixed roles
  - Software Service provider (its natural role)
  - Must manage allocation of hardware resources
  - Must manage OS images, their upgrades, and their base software
  - Must build their own service management strategy
    - Monitoring mechanisms
    - Upgrade mechanisms

# 4.3. Service Evolution:
# g) Platform as a Service (PaaS)

- Ideally, promises to take away extraneous tasks from SaaS providers
  - Still in infancy, though
- Purports to be the equivalent of an OS
  - Specifies a Service Model on which to base specification of SaaS, and development of their software components.
  - Includes the following aspects
    - Configuration and lifetime management model (including dependency expressions)
      - Composition, Configuration, Deployment and Upgrade mechanisms
    - Performance model
      - Automatic monitoring of relevant parameters
      - Expression of elasticity points
      - Automatic reconfiguration under varying load

# 4.4. Summary

- Cloud Computing: Ideal layering

# 4.4. Summary

- Cloud Computing is all about efficiency and easiness:
  - Efficient sharing of resources
    - Consume only what one needs
    - Pay only for what is used
  - Easy adaptation to a varying population of users
  - Easy ways to develop and provide a service
- Three layers of cloud services are identified:
  - Software as a Service
    - The actual goal is to provide these to a large user population
  - Platform as a Service
    - Extremely desirable to automate management of resources for SaaS and ease creation and deployment of services.
  - Infrastructure as a Service
    - Enables SaaS elasticity
- From the user's point of view CC feels like going back to the Mainframe era

# Index

1. Concept of Distributed System
2. Relevance
3. Application Areas
4. Back to the beginning: Cloud Computing
5. Programming Paradigms
6. Conclusions: No computing without a Network

# 5. Programming Paradigms

- A prevalent way to organize a distributed system is to make each process a "server"
  - Receives requests, processes them, sends back answers
- Servers, themselves, request services from other servers
  - They may need such services to satisfy a request they receive
- To be able to scale, servers must not BLOCK while serving request
  - Should be able to accept other requests
- Two programming paradigms exist:
  - Multi-threaded programming
    - State-sharing concurrent servers
  - Asynchronous programming
    - Asynchronous single-threaded servers

# 5.1. Concurrent, state-sharing servers

- Multi-threaded concurrent programs
  - Each request is handled in its own thread
  - All threads share a global state
  - Concurrency-control mechanisms are used to implement atomicity
- Advantages
  - Threads may block waiting for requests from the server to complete, without blocking the server
- Disadvantages
  - Multi-threaded programming has its own overheads
    - Need to support concurrency control constructs
  - It turns out that shared memory concurrent programming is
    - Hard to do well
    - Hard to reason about how it works
- Prevalent Environments:
  - Java
  - .NET

# 5.2. Async Servers

- Async programming *aka* event-driven programming.
  - Closely matches the guard-action program model
  - Async programs have many activities, but…
    - State is never <u>concurrently shared</u> among the executing activities
- "Events" are the "guards".
- Actions are established as callbacks of events
  - Need to dynamically built actions/guards to facilitate programming
  - When building actions, programming language mechanisms are used to easily establish the state to be affected by the action
    - Reduce complexity of "preparing" state to link internal actions
- Actions ready for execution placed on a "turn queue"
  - Scheduled actions executed in FIFO order of the queue

# 5.2. Async Servers

- Advantages
  - Shared state handling complexity greatly diminishes
    - Still, careful on handling of the turn queue to avoid surprises
  - Less overhead, as no multi-threading environment is supported
    - Better ability to scale
  - Close match to how a distributed system actually work: event-driven
    - Easier to reason about what is going on
- Disadvantages
  - Proper state-handling is necessary when building actions
  - Needs all environment to be async, not just IPC
    - OS services need to be async too, to avoid blocking
- Prevalent environments with built-in support in language
  - Nodejs
  - Async .NET

# Index

1. Concept of Distributed System
2. Relevance
3. Application Areas
4. Back to the beginning: Cloud Computing
5. Programming Paradigms
6. Conclusions: No computing without a Network

# 6. Conclusions

- Networked Systems are Distributed Systems
  - Most computing nowadays is networked
    - Thus, distributed
  - Proper design and implementation requires mastering aspects of concurrent programming and properties of the architecture
- Rich set of application areas already exploited
- Important trend of Cloud Computing as a logical endpoint in the evolution of computing
  - Driven by efficiency in resource usage
  - Pay-as-you-go model of access
  - Elasticity and scalability
- Two programming paradigms for distributed service development:
  - Concurrent (i.e., multi-threaded) servers
    - Should deal with race conditions. Threads may be blocked.
  - Asynchronous servers
    - Event-driven. Easily scalable.