Imperial College London

Computer Networks and Distributed Systems

Part 2.1 – Distributed Systems Architecture

Course 527 – Spring Term 2015-2016

Emil C Lupu and Daniele Sgandurra

e.c.lupu@imperial.ac.uk, d.sgandurra@imperial.ac.uk

Recommended Books (for DS)

"Distributed Systems: Concepts and Design", George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair Addison-Wesley, 2005 (5th Ed.)

"Distributed Systems: Principles and Paradigms" Andrew S Tanenbaum, Maarten Van Steen, Pearson Ed. (2nd Ed.)

Acknowledgements:

- Distributed Systems based on material by Morris Sloman
- Some of the figures and images from external sources.

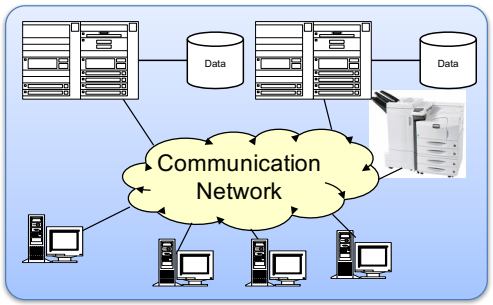
Contents

Definition

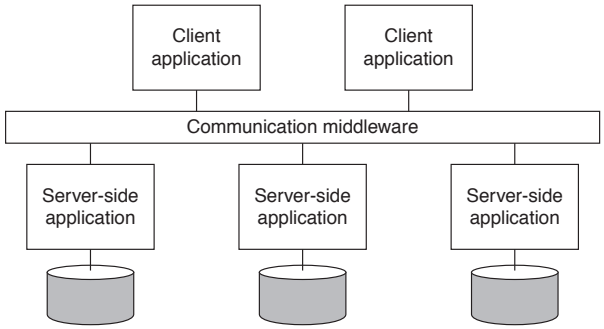
A distributed system consists of a collection of autonomous computers interconnected by a computer network and equipped with distributed system software to form an integrated computing facility.

Components interact and cooperate to achieve a common goal.

Processes co-ordinate by means of **messages** transferred over a communication network.



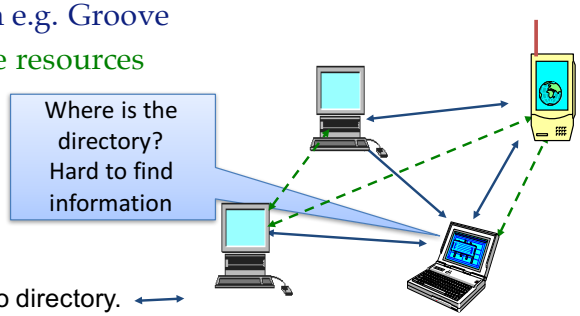
Enterprise Systems



Remote Procedure Calls
Message Oriented Middleware
Publish/Subscribe Systems

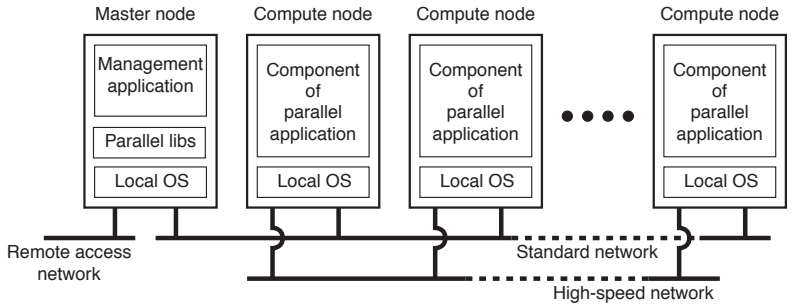
Peer-to-peer (P2P)

Very large scale – potentially millions of users
Share processing eg Seti@home, United Devices, Avaki, Akamai
'Share' files eg Gnutella, Kazaa
Collaboration e.g. Groove
How to locate resources



Publish and query to directory. ←→
Get data from peers ←- - →

Computing Clusters



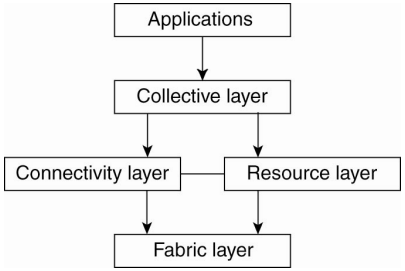
Generally used for parallel programming.
Off-the-shelf computers interconnected by a high speed network.
Examples: Beowolf, CoW

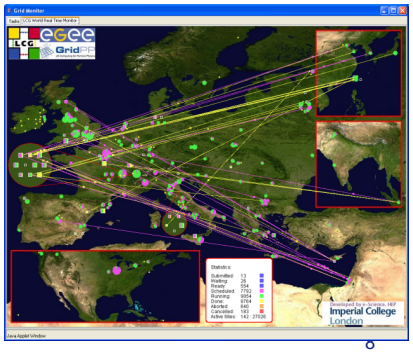
Grid Computing

Distributing computation and data across several sites.

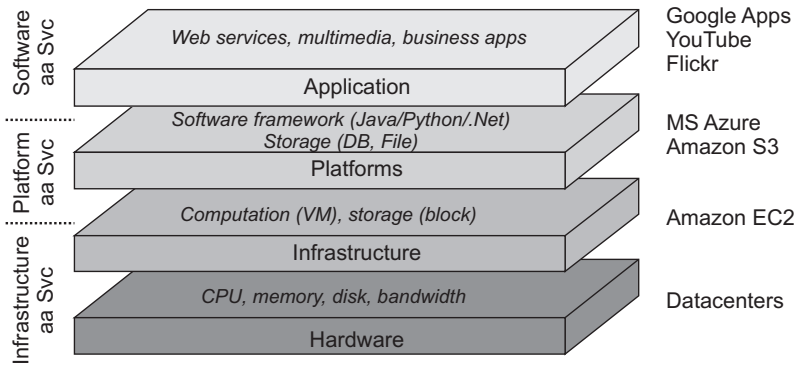
Increasingly based on Web Service Architectures

Examples: OGSA, Globus, OntoGrid





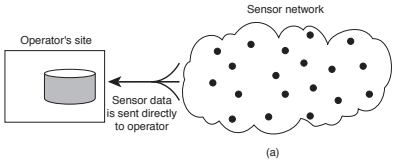
Cloud Computing

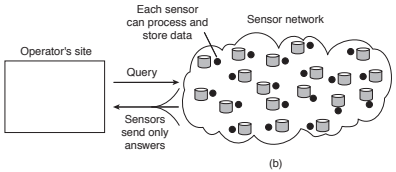


Virtualisation of HW and SW infrastructure.

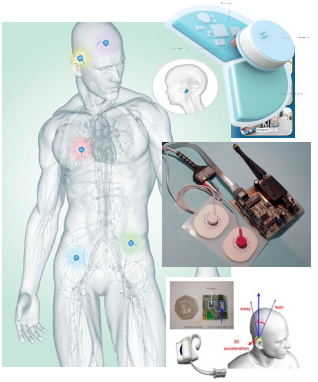
(Distributed) Pervasive Systems IoT

Wireless Sensor Networks

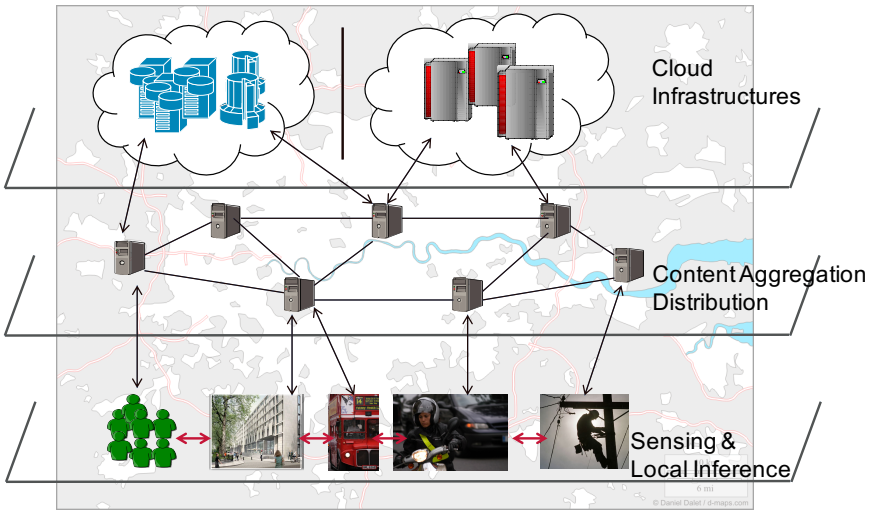




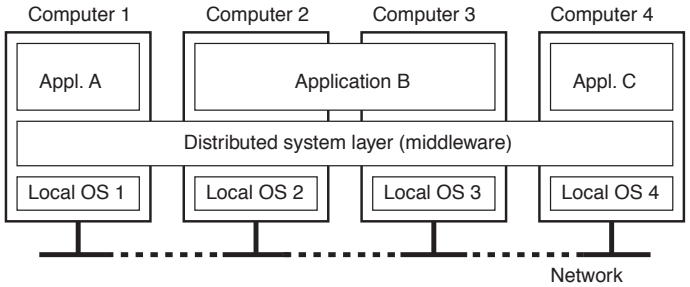
Body Sensor Networks



Large Scale Infrastructures



So What is a Distributed System?



Abstracting from the specificities of every node to consider the computing environment as a single coherent system.

Requires: interaction abstractions, middleware and services

Characteristics/Advantages

Resource sharing: remote access to shared facilities

Fault tolerance: replication can remove failures

Concurrency: reduce response time by local processing, improve throughput by parallelism

Openness: vendor independence via clearly defined interfaces and use of standards

Scalability: via multiple processors and multiple networks

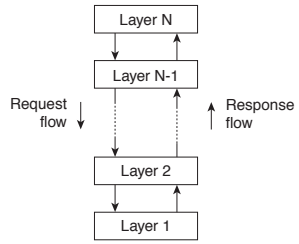
Modularity: simpler design, installation & maintenance

Flexibility: incremental change of function & adaptation to new requirements

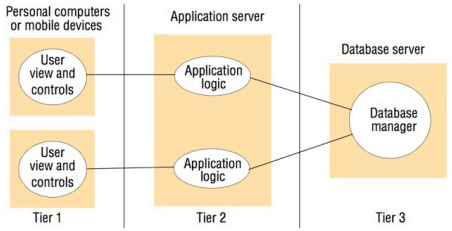
Reflect application distribution

But no global time: difficult to support causality and consistency

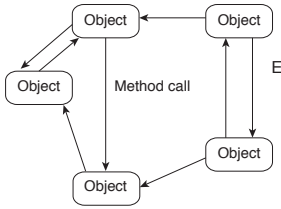
Architectural Styles



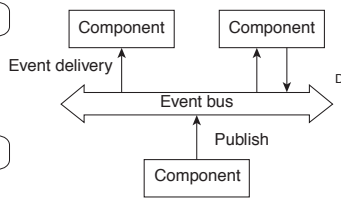
Layers



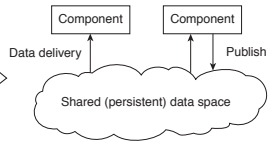
Tiers



Peers (Objects, Proc.,…)

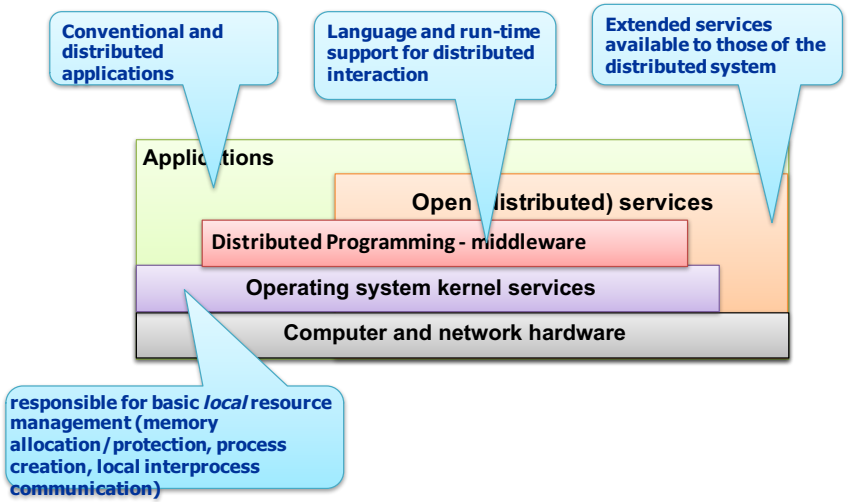


Event Based



Shared memory based

Basic software structure



Open services:

- support the introduction of new services
- provide access to distributed services, including the coordination required for remote resource use (sharing, protection, synchronisation, recovery....)
- e.g. Jini resource discovery,

Distributed programming support:

- supports interaction (such as remote procedure call) for conventional languages and support for special purpose languages.
- e.g. Java RMI, RPC

Distribution Transparencies

It is often useful to hide distribution from the user – design goals which can be difficult to achieve

Access	uniform access whether local or remote
Location	access without knowledge of location ie location independent naming
Concurrency	sharing without interference – requires synchronisation
Replication	hides use of multiple components eg for fault tolerance
Failure	concealment of faults by replication or recovery
Migration	hides movement of components eg for load balancing
Performance	use of scheduling and reconfiguration to hide performance variations.
Scaling	permits expansion without changing system structure or algorithms
Heterogeneity	transforming information between different representations

Open Distributed Processing – Viewpoints

Viewpoint = abstract representation of a system NOT phases in lifecycle model

Enterprise Viewpoint

- Overall goals, policies & organisational structure
- Roles & activities within organisation(s)
- Policies & constraints regarding cross-organisation interactions
- Community: configuration of objects established to meet an objective – specifies roles, relationships and policies

Information Viewpoint

- Modelling of information structures, information flows and knowledge representation
- Includes constraints on data
- No distinction between manual & automated information processing

Computational Viewpoint

- Programming functions – IPC, object interfaces
- Application program structuring – independent of computer system on which it will run
- No distinction between processing & storage object
- Includes configuration – object instantiation and bindings.

Engineering Viewpoint

- OS, communication system, database – implementation issues
- Provision of transparency mechanisms – fault tolerance, persistence etc.
- Processors & networks are visible

Technology Viewpoint

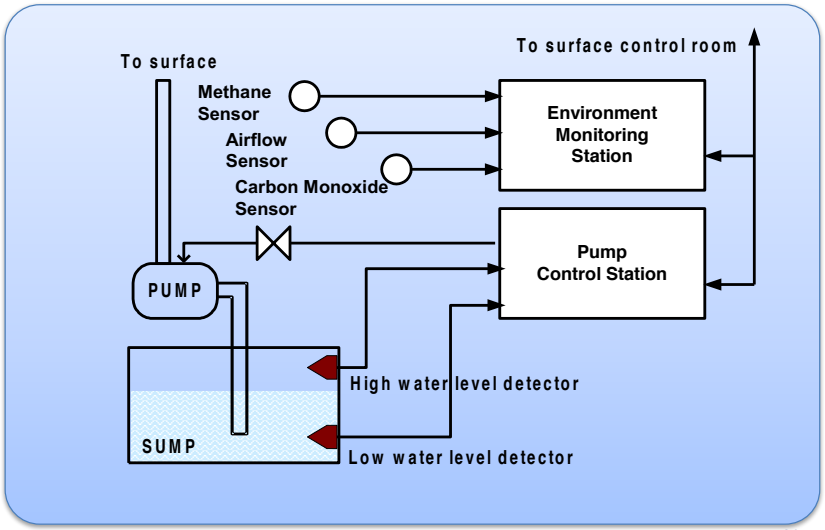
- Realised components from which distributed systems are built.
- Particular OS (Unix, Windows), protocols (FTP, TCP/IP), processors (Intel, sparc, ARM)

Example: Pump for Mine Drainage

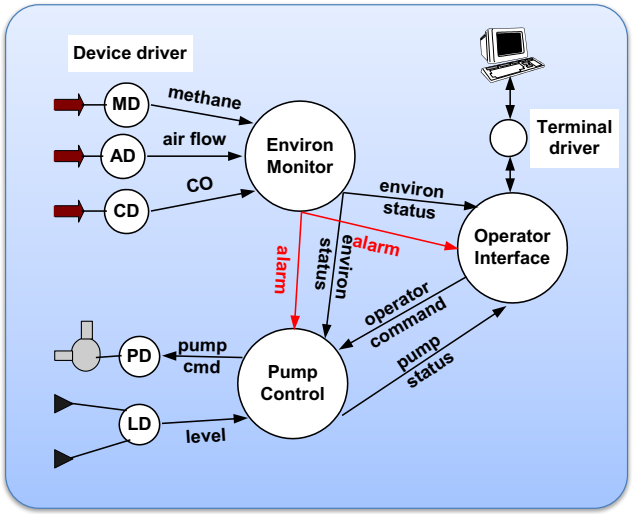
The pump is situated underground in a coal mine, and so for safety reasons it must not be started or continue running when the percentage of methane in the atmosphere exceeds a set safety limit. The pump controller obtains information on methane levels by communicating with a nearby environment monitoring station. As well as methane, this station also monitors carbon monoxide and airflow velocity. The environment monitoring station provides information to the surface and other plant controllers as well as to the pump controller.

Once start has been enabled by a command from the surface, the pump runs automatically controlled by the water level as sensed by the high and low level detectors. Detection of high level causes the pump to run until low level is reached. The surface may deactivate the pump with a stop command, and also query the status of the pump.

Pump System Overview



Pump Control Schematic



Data Flow Diagrams

Component Processes describe the functions of the system

Data flows = data type + direction

Data Dictionary for Pump System

```
pump cmd     = (on, off)
level        = (high, low)
methane      = real
airflow      = real
alarm        = signal

environ status   = methane + airflow + CO

operator cmd     = (start, stop, status)

pump status =  (stopped, lowstop, methanestop,
running)
```


Distributed System Design Approach

Enterprise View

- Specify requirements and identify interactions with the environment
- Identify main processing components – processes or threads of control
- Assume 1 process per device

Information View

- Identify data flows – direction and data types (dictionary)
- Ignore how interactions are initiated or types of interaction primitives

Computation View

- Decide on interaction primitives
- Decide on control flow i.e. whether data is pushed or pulled
- e.g. whether controllers are polled or event driven
- Specify component interfaces
- = interactions + signatures (parameter types)
- Specify component functions in terms of outline code and data structures

Engineering View

- Optimise and allocate to physical nodes

Architecture Summary

What are distributed systems – definition
Why are they of interest – potential benefits
Where are they used – applications

Architecture

- Viewpoint decomposition
- Architectural Style

Main Design aspects