

Distributed Systems COMP28112

Dr. Sandra Sampaio



Introduction to Distributed Systems

Sandra Sampaio

2



The Evolution of Computer Systems

- From 1945 to 1985
 - Computers were large and expensive.
 - It was not possible to connect computers, so they operated independently from one another.
- From 1985
 - Advances in technology began to change that situation, including:
 - the development of powerful microprocessors,
 - the invention of high-speed computer networks, and
 - the miniaturization of computer systems.
- The result of these technologies is that it is, now, possible and relatively easy to put together a computing system composed of large numbers of networked computers, large or small.

As these computers are generally dispersed, they are said to form a distributed system.



Examples of Distributed Systems

- The Web over the Internet
 - Documents, email, media, commerce, etc.
- Mobile telephony
 - Calls, texts, location, sensing, etc.
- Electronic funds transfer
 - Banking, credit cards, etc.
- Instant messaging
- Video conferencing
- Home entertaining systems
- Global positioning systems
- Etc.



Why Distributed Systems?

- To share resources:
 - One publisher, many beneficiaries!
- To bind customers and suppliers
 - “Go ahead, tell us what you want, we’ll sell it!”
 - “This is what we want, if you have it, we’ll buy it!”
- To allow us to do things we could not otherwise do, due to performance, scalability, reliability and availability issues.
 - Performance issues: e.g., if using 1 computer, it takes 60 minutes, I’ll use 100 computers and it will take 0.6 minutes!
 - Scalability issues: e.g., if there’s 10 times more to do in the 0.6 minutes I have, I’ll use 1,000 computers, then, if it things quieten down again, I’ll go back to using 100 computers!
 - Reliability and availability issues: e.g., if 1% of computers fails every day, add an extra 1% to keep the 0.6 stable and on!



In other words:

- To make continuously-evolving, remote resources accessible for **sharing**.
- To open proprietary processes to external interaction in order to foster **cooperation**.
- To achieve better **performance/cost** ratios.
- To **scale** effectively and efficiently if **demand for resources** changes significantly.
- To **scale** through modular, incremental **expansion and contraction**.
- To achieve high levels of **reliability** and **availability**.



What is a Distributed System?

*“A distributed system is a collection of autonomous **computing elements** that appears to its users as a **single coherent system**”*

Where:

- A computing element or a **node**, can be either a hardware device or a software process.
 - Note that a node can be anything from a high-performance mainframe computer to a small device in a sensor network. Also, nodes can be interconnected in anyway.
- In a single coherent system, users (i.e., people or applications) believe they are dealing with a single system.
 - Note that, for this to be possible, the **autonomous nodes** need to collaborate. The way in which this collaboration is established represents the most fundamental property that distinguishes between different distributed systems.

As a consequence, each node will have its own notion of time.



Characteristics of Distributed Systems

- Distributed systems can also vary in:
 - Size (e.g., from a few to millions of computers).
 - The way in which nodes are interconnected (e.g., via a wired, wireless or a hybrid (a combination of both) network).
 - The way in which node membership is handled (e.g., by being open or closed group when allowing new nodes to join).
- Note that, although nodes can act independently from one another, they cannot ignore one another. Otherwise, there is no point in putting them to compose the same system!
- Nodes are programmed to achieve common goals, and they do this by **exchanging messages** with each other.
- To appear to users as a single coherent system, **distribution transparency** is an important goal of distributed systems.

This is a concept that we will discuss in more detail later.



In addition, in distributed systems:

- Computation is concurrent
 - There is more than one application running.
- There is no shared state.
 - There is no single global clock that all programs can agree to follow.
- Failures occur and we may not know or be told about them.

How can we tell whether a remote program crashed or is just taking long?


How do we make programs synchronize their actions?

How do we make sure that there is no mishap?



- Communication events have non-negligible duration.
 - Communication costs may become more significant than processing time.
- Components may exchange data at variable rates.
 - Uncertainty as to when and how long is always present.
- Different components process data at different rates.
 - Asynchrony (i.e., non-aligned timelines) is unavoidable.

What can we do when
we need information?



Fallacies about Distributed Systems

“Distributed systems differ from traditional software because components are dispersed across a network. Not taking this dispersion into account during design time is what makes so many systems needlessly complex and results in flaws that need to be patched later on”.

(Distributed Systems, M. van Steen and A. S. Tanenbaum, Third edition, Chapter 1)

- When working at Sun Microsystems, *Peter Deutsch* formulated the following false assumptions that are commonly made when developing a distributed application for the first time:
 1. The network is reliable
 2. The network is secure
 3. The network is homogeneous
 4. The topology does not change
 5. Latency is zero
 6. Bandwidth is infinite
 7. Transport cost is zero
 8. There is one administrator



Fallacy 1: The Network is Reliable

- The way in which nodes of a distributed system are linked (a.k.a. network topology) may vary.
- A distributed system is composed of different and varied (i.e., heterogeneous) resources ([software](#) and [hardware](#)), which are deployed by different people/organisations.
- In essence, the result is a network of networks, each managed with local scope by a different group of administrators
- As a consequence, no (sufficiently complex to be useful) network of networks is reliable.

Example: Lack of software reliability – need for reliable message exchange between nodes (e.g., functionality for retrying messages, acknowledging messages, verifying message integrity, etc.)

Example: Hardware associated power failures – network switches have a mean time between failures (e.g., the route between you and the server you get data from). Need for weighting the risks of failure versus the required investment to build redundancy - a trade-off!



Fallacy 2: The Network is Secure

“In case you landed from another planet, the network is far from being secured”

(common wisdom)

- The Implications:
 - You may need to build security into your applications from Day 1.
 - As a result of security considerations, you might not be able to access networked resources, different user accounts may have different privileges, and so on...



Fallacy 3: The Network is Homogeneous

- A homogeneous network today is the exception, not the rule!
 - Even a home network may connect a Linux PC and a Windows PC.
- Implications:
 - Interoperability will be needed.
 - And so, the use of standard technologies (not proprietary protocols), such as XML (a W3C recommended general-purpose markup language) or JSON will be necessary.

Homogeneous = of the same kind; uniform.

A **markup language** combines text and extra information about the text – designed to facilitate the sharing of data across different information systems. Its drawback? It's slow...



Fallacy 4: Topology doesn't Change

- The topology doesn't change as long as we stay in the lab!
- In the wild, servers may be added and removed often, clients (laptops, wireless ad hoc networks) are coming and going: the topology is changing constantly.
- The implications:
 - Do not rely on specific endpoints or routes.
 - Abstract from the physical structure of the network, by (using the most obvious example) using DNS names as opposed to IP addresses (which may vary) for referring to an endpoint.

Network topology is the way in which nodes of a distributed system are linked.

DNS (Internet Domain Name System) is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It is commonly used to translate more readily memorized domain names to the numerical IP addresses needed for locating and identifying computer services and devices with the underlying network protocols.



Fallacy 5: Latency is Zero

- The minimum round-trip time between two points on earth is determined by the **maximum speed of information transmission**: the speed of light.
 - At 300,000 km/sec, it will take at least 30msec to send a **ping** from Europe to the USA and back.
- The implications:
 - You may think all is ok if you deploy your application on LANs, but you should strive to make as few calls over the network as possible (and transfer as much data out in each of these calls).

Network latency is a time measure of how long it takes for data to move from one place to another.

Ping is a computer network administration software utility that measures the round-trip time for messages sent from the originating host to a destination computer that are echoed back to the source.



Fallacy 6: Bandwidth is Infinite

- Bandwidth trends show that it constantly grows, but so does the amount of information we are trying to squeeze through it!
 - E.g., videos, audio, photographs, messages, ...
- To avoid congestion and increase connection throughput, the loss of data packets being transmitted from one point to another can be performed.
 - To avoid packet loss, we may want to use larger packet sizes.
- The implications:
 - Data compression; try to simulate the production environment to get an estimate for your needs.

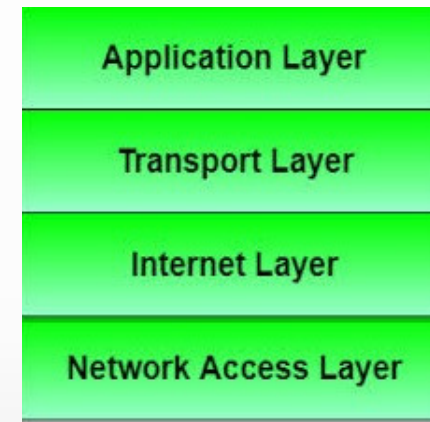
Bandwidth is a measure of how much data it is possible to transfer over a period of time (may be measured in bits/second).

Throughput is a measure of how much data is successfully transferred from source to destination within a given timeframe. In other words, it is used to measure how many packets arrive at their destinations successfully. It can be measured in bits/second.



Fallacy 7: Transport Cost is Zero

- Going from the application layer to the transport layer (2nd highest in the five layer TCP/IP reference model for network communication) is not free:
 - Information needs to be serialised (by marshalling) to get data onto the wire.
 - The cost (in terms of money) for setting and running the network is not zero.
Have we leased, for instance, the necessary bandwidth?



TCP/IP Reference Model
Figure source: studytonight.com

Marshalling is the process of transforming the memory representation of an object to a data format suitable for storage or transmission.



Fallacy 8: There is One Administrator

- Unless we refer to a small LAN, there will be different administrators associated with the network with different degrees of expertise.
- Might make it difficult to locate problems (*is it their problem or ours?*)
- Coordination of upgrades: *will the new version of MySQL work as before with Ruby on Rails?*
- **Don't underestimate the 'human' ('social') factor!**



Distributed Systems Organisation

- To assist the development of distributed applications, distributed systems are often organized to have a separate layer of software that is logically placed on top of the respective operating systems of the computers that are part of the system.
- This separate layer is called middle layer and offers each application the same interface as well as the following:
 - Facilities for inter-application communication.
 - Security services.
 - Support for transaction management.
 - Recovery from failures.
 - Support for [Web services composition](#).
 - Etc.

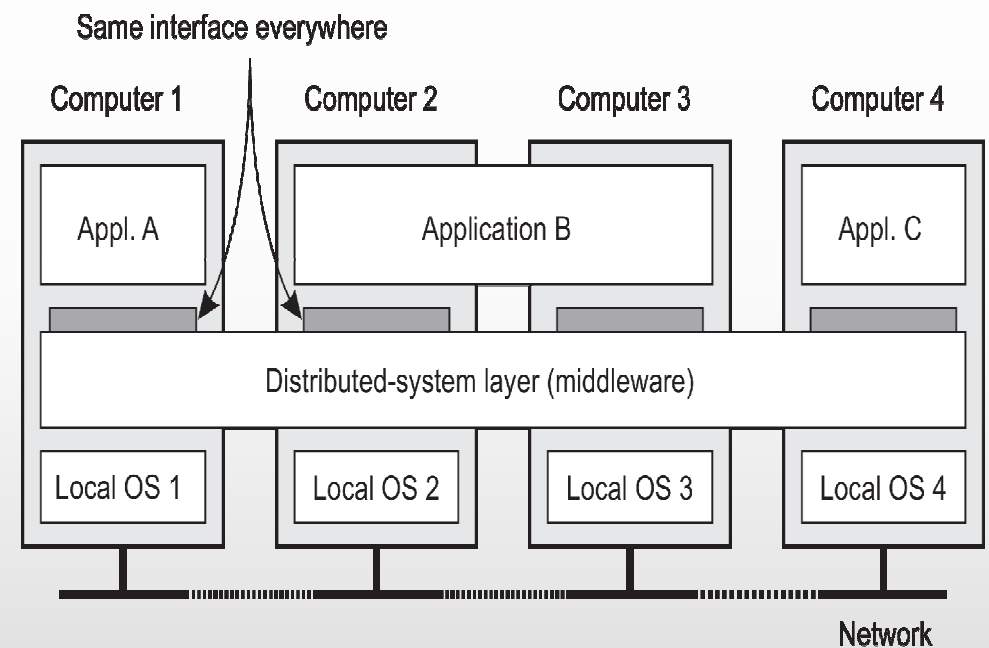


Figure source: Distributed Systems, M. van Steen and A. S. Tanenbaum, Third edition.

It is becoming increasingly common to develop new applications by taking existing programs and gluing them together.



Designed for Transparency

Transparency forms	Description
Access	Hide differences in data representation and how an object is accessed.
Location	Hide where an object is located.
Relocation	Hide that an object may be moved to another location while in use.
Migration	Hide that an object may move to another location.
Replication	Hide that an object is replicated.
Concurrency	Hide that an object may be shared by several independent users.
Failure	Hide the failure and recovery of an object.

An object can be a resource or a process.

