Distributed computing
“Introduction to distributed computing”

Mathieu Delalandre
University of Tours, Tours city, France
mathieu.delalandre@univ-tours.fr
February 5, 2019
Introduction
to distributed computing

1. Introduction to distributed systems
   1.1. Definitions, application domains and motivations
   1.2. Relations to parallel systems
   1.3. Goals, challenges and trends

2. Architecture, design, communication and computing
   2.1. The layered model and design issues
   2.2. Primitives for communication
   2.3. Distributed computing, definitions and challenges
Introduction to distributed systems

“Definitions” (1)

“A distributed system is a collection of independent computers that appear to the user as a single computer.” [A. Tanenbaum]

“A distributed system is one in which components located at networked computers communicate and coordinate their actions by passing messages. This definition leads the following characteristics of distributed systems: concurrency of components, lack of global clock and independent failures of components.” [G. Coulouris]

“A distributed system is one on which I can’t do my work due to a computer that has failed and I never heard of.” [L. Lamport]
A distributed system has the following features:

**No common physical clock:** this is an important assumption because it introduces the element of “distribution” in the system and gives rise to the inherent asynchrony amongst the processors.

**No shared memory:** this is a key feature that requires message-passing for communication. This feature implies the absence of the common physical clock.

**Geographical separation:** the geographically wider apart that the processors are, the more representative is the system of a distributed system.

**Autonomy and heterogeneity:** the processors are “loosely coupled” in that they have different speeds and each can be running a different operating system. They are usually not part of a dedicated system, but cooperate with one another by offering services or solving a problem jointly.
Introduction to distributed systems
“Application domains”

The application domains are:

The information society: the growth of the W3 as a repository of information and knowledge, the development of web search engines such as Google or Yahoo.

Finance and commerce: the growth of the eCommece as exemplified by companies such as Amazon and eBay, and underlying payments technologies (e.g. PayPal).

Creative industries and entertainment: the emergence of online gaming as a novel and highly interactive form of entertainment with Massively Multiplayer Online Games (MMOGs) (e.g. WoW).

Education: the emergence of e-learning through for example web-based tools such as virtual learning environments (e.g. Moodle).

Transport and logistic: the use of location technologies such as GPS in route finding systems and more general traffic management systems (e.g. Waze).

Science: the emergence of the Grid as a fundamental technology for eScience, including the use of complex networks of computers to support the storage (e.g. SETI project).

Healthcare: the growth of health informatics as a discipline with emphasis on online electronic patient records and related issues of privacy (e.g. smartwatch)

Environmental management: the use of (networked) sensor technology to both monitor and manage the natural environment (e.g. smart farming).
Introduction to distributed systems
“Motivations”

The motivations for using a distributed system are:

**Inherently distributed computation:** in many applications such as money transfer in banking, or reaching consensus among parties that are geographically distant, the computation is inherently distributed.

**Resource sharing:** resources such as peripherals, databases, libraries, etc. cannot be fully replicated at all the sites because it is often neither practical nor cost-effective. Further, they cannot be placed at a single site because access to that site might prove to be a bottleneck. Therefore, such resources are typically distributed across the system.

**Access to geographically remote data and resources:** special resources such as supercomputers exist only in certain locations, and to access such supercomputers, users need to log in remotely.

**Enhanced reliability:** a distributed system has the inherent potential to provide increased reliability because of the possibility of replicating resources and executions, as well as the reality that geographically distributed resources are not likely to crash/malfunction at the same time under normal circumstances.

**Increased performance/cost ratio:** by resource sharing and accessing geographically remote data, the performance/cost ratio is increased. Such a configuration provides a better performance compared to special parallel machines.

**Scalability:** as the processors are usually connected by a wide-area network, adding more processors does not pose a direct bottleneck for the communication network.

**Heterogeneity and incremental expandability:** heterogeneous processors may be easily added into the system without affecting the performance, as long as those processors are running the same middleware algorithms. Similarly, existing processors may be easily replaced by other processors.
Introduction
to distributed computing

1. Introduction to distributed systems
   1.1. Definitions, application domains and motivations
   1.2. Relations to parallel systems
   1.3. Goals, challenges and trends

2. Architecture, design, communication and computing
   2.1. The layered model and design issues
   2.2. Primitives for communication
   2.3. Distributed computing, definitions and challenges
Flynn’s taxonomy identifies four processing modes. It is instructive to examine this classification to understand relation of distributed systems to parallel systems:

**Single instruction stream, single data stream (SISD):** this mode corresponds to the conventional processing in the von Neumann paradigm with a single CPU, and a single memory unit connected by a system bus.

**Single instruction stream, multiple data stream (SIMD):** this mode corresponds to the processing by multiple homogenous processing units which execute in lock-step on different data items. It targets applications that involve operations on large arrays and matrices, such as scientific applications.

**Multiple instruction stream, single data stream (MISD):** this mode corresponds to the execution of different operations in parallel on the same data. This is a specialized mode of operation with limited but niche applications, e.g., visualization.

**Multiple instruction stream, multiple data stream (MIMD):** in this mode, the various processing units / processors execute different code on different data. This is the mode of operation in distributed systems as well as in the vast majority of parallel systems.
Flynn’s taxonomy identifies four processing modes. It is instructive to examine this classification to understand relation of distributed systems to parallel systems:

**Single instruction stream, single data stream (SISD):** This mode corresponds to the conventional processing in the von Neumann paradigm with a single CPU, and a single memory unit connected by a system bus.

**Single instruction stream, multiple data stream (SIMD):** This mode corresponds to the processing by multiple homogenous processing units which execute in lock-step on different data items. It targets applications that involve operations on large arrays and matrices, such as scientific applications.

**Multiple instruction stream, single data stream (MISD):** This mode corresponds to the execution of different operations in parallel on the same data. This is a specialized mode of operation with limited but niche applications, e.g., visualization.

**Multiple instruction stream, multiple data stream (MIMD):** In this mode, the various processing units/processors execute different code on different data. This is the mode of operation in distributed systems as well as in the vast majority of parallel systems.
Relations to parallel systems (3)

<table>
<thead>
<tr>
<th>Processing mode</th>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Clock</td>
<td>Communication</td>
</tr>
<tr>
<td>bus-based multiprocessors</td>
<td>synchronous</td>
<td>bus</td>
</tr>
<tr>
<td>switched multiprocessors</td>
<td>MIMD</td>
<td>switch</td>
</tr>
<tr>
<td>network operating system</td>
<td>asynchronous</td>
<td>LAN</td>
</tr>
<tr>
<td>distributed system</td>
<td>asynchronous</td>
<td>WAN</td>
</tr>
</tbody>
</table>

synchronous systems

e.g. The OMEGA switching network

each switch has two inputs/outputs
each switch works as a router
for N memories (M) and N CPUs (C),
the OMEGA requires \( \log_2(N) \) switches
### Relations to parallel systems (4)

<table>
<thead>
<tr>
<th>Processing mode</th>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Clock</td>
<td>Communication</td>
</tr>
<tr>
<td>bus-based multiprocessors</td>
<td>synchronous</td>
<td>bus</td>
</tr>
<tr>
<td>switched multiprocessors</td>
<td>MIMD</td>
<td>switch</td>
</tr>
<tr>
<td>network operating system</td>
<td>asynchronous</td>
<td>LAN</td>
</tr>
<tr>
<td>distributed system</td>
<td>asynchronous</td>
<td>WAN</td>
</tr>
</tbody>
</table>

**ipc**
- Interoperability
- Shared data
- Time synchronization
- System coordination

<table>
<thead>
<tr>
<th>Network operating systems</th>
<th>Distributed systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
Introduction
to distributed computing

1. Introduction to distributed systems
   1.1. Definitions, application domains and motivations
   1.2. Relations to parallel systems
   1.3. Goals, challenges and trends

2. Architecture, design, communication and computing
   2.1. The layered model and design issues
   2.2. Primitives for communication
   2.3. Distributed computing, definitions and challenges
Goals, challenges and trends (1)

Goals and challenges …

**Accessibility:** the main goal of distributed systems is to make it easy, for the users and applications, to access remote resources and to share them in a controlled and efficient way.

**Security:** as connectivity and sharing increase, security is becoming increasingly important.

**Failure handling:** computer systems sometimes fail that could produce incorrect results of programs. These failures are partial i.e. some components fail while others continue.

**Concurrency:** there is possibility that several applications will attempt to access a shared resource at the same time. The process that manages a shared resource could take one request at a time. Therefore, distributed components generally allow multiple application requests to be processed concurrently.

**Heterogeneity:** the Internet enables users and applications to access and run programs over an heterogeneous collection of computers and networks.
Goals, challenges and trends (2)

Goals and challenges …

**Openness:** the openness in a distributed system is the characteristic that determines whether the system can be extended and re-implemented in a various way. It is characterized by the degree to which new resources-sharing and services can be added and made available for use by a variety of applications.

**Scalability:** a system is described as scalable if it will remain effective considering a significant increase in the number of resources and users.

**Transparency:** a distributed system that is able to present itself to users and applications as if it were only a single computer system is said transparent.

**Pitfalls:** developing a distributed system differs from traditional programming because components are dispersed across network. No tacking this dispersion into account could result in mistakes.
Goals, challenges and trends (3)

Trends …

**Pervasive networking and the modern internet**: the modern internet is a vast interconnected collection of computer networks of many different types increasing all the time (WiFi, WiMAX, Bluetooth, etc.). The net results is that networking has become a pervasive resource and devices can be connected at any time and in any place.

**Distributed multimedia systems**: the benefits of distributed multimedia services are considerable in that a wide range of new applications can be provided (television broadcast, video-on-demand, music libraries, audio and video conferencing, etc.). The crucial characteristic of continuous media is that they include a temporal dimension and need to preserved their real-time relationships.

**Distributed computing as utility**: with the increasing maturity of distributed infrastructures, a number of companies are promoting the view of distributed resources as a commodity or utility (i.e. as water or electricity). This model applies both physical resources (storage, backup, distributed computation, bandwidth, etc.) and logical (email, distributed calendar, business apps, etc.). The term cloud computing is used to capture this vision of computing as utility.

**Ubiquitous computing** is a post-desktop model of human-computer interaction in which information processing has been thoroughly integrated into everyday objects and activities. More formally ubiquitous computing is defined as "machines" that fit the human environment instead of forcing humans to enter theirs.
Introduction to distributed computing

1. Introduction to distributed systems
   1.1. Definitions, application domains and motivations
   1.2. Relations to parallel systems
   1.3. Goals, challenges and trends

2. Architecture, design, communication and computing
   2.1. The layered model and design issues
   2.2. Primitives for communication
   2.3. Distributed computing, definitions and challenges
The layered model and design issues (1)

The layered model at the distributed system level …

**Users:** could share a same computer (through session / terminal).

**Applications, services:** designed to help the user to perform a singular or multiple related specific tasks (e.g. office, programming toolkit, web browser, etc.).

**Middleware** is an application that logically lives (mostly) in the application layer, but which contains many general-purpose protocols that warrant their own layers, including the session and presentation layers. Some of the middleware protocols could equally belong to the transport protocol.

**Table:**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocols, e.g.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>HTTP, FTP, SMTP, etc.</td>
</tr>
<tr>
<td>Presentation</td>
<td>SSL, CORBA, etc.</td>
</tr>
<tr>
<td>Session</td>
<td>RPC, RMI, etc.</td>
</tr>
<tr>
<td>Transport</td>
<td>TCP, UDP, etc.</td>
</tr>
</tbody>
</table>

**Diagram:**

- **OSI Model (Open System Interconnection)**
  - Application
  - Presentation
  - Session
  - Transport
  - Network
  - Data Link
  - Physical

- **Protocols, e.g.:**
  - Application: HTTP, FTP, SMTP, etc.
  - Presentation: SSL, CORBA, etc.
  - Session: RPC, RMI, etc.
  - Transport: TCP, UDP, etc.
Two main approaches for middleware design exist:

**Inter-Process Communication (IPC) and synchronization:** IPC is related to a set of methods for the exchange of data among multiple threads and/or processes. Processes may be running on one or more computers connected by a network. The methods of IPC may vary based on the bandwidth and latency of communication, and the type of data being communicated. Synchronization refers to the idea that multiple processes are to join up or handshake at a certain point, so as to reach an agreement or commit to a certain sequence of action.

**Distributed objects / components and remote invocation** is concerned with programming models for distributed applications. That is, these applications are composed of cooperating programs running in several different processes. Such programs need to be able to invoke operations in other processes, often running in different computers.
Introduction to distributed computing

1. Introduction to distributed systems
   1.1. Definitions, application domains and motivations
   1.2. Relations to parallel systems
   1.3. Goals, challenges and trends

2. Architecture, design, communication and computing
   2.1. The layered model and design issues
   2.2. Primitives for communication
   2.3. Distributed computing, definitions and challenges
Primitives for communication (1)

Message passing and primitives for communication: all the communication in a distributed system is based on message passing. Message passing communication is about the mechanisms used to send and to receive messages. These mechanisms involves a set of primitives used for communication (e.g. UDP, TCP, RPC, MPI, etc.).
Send and receive primitives: in a message passing communication, a message is sent and received by explicitly executing the send and receive primitives, respectively. The following are some definitions of primitives for communication:

Send primitive: the send primitives has at least two parameters (a) the destination (b) the buffer in the user space, containing the data to be sent.

Receive primitive: similarly, the receive primitive has at least two parameters (a) the source from which the data is to be received (2) the user buffer into which the data is to be received.
Primitives for communication (3)

Send and receive primitives: in a message passing communication, a message is sent and received by explicitly executing the send and receive primitives, respectively. The following are some definitions of primitives for communication:

**Blocking primitives:** a primitive is blocking if control returns to the invoking process after the processing for the primitive (whether in synchronous or asynchronous mode) completes.

**Non-blocking primitives:** a primitive is non-blocking if control returns back to the invoking process immediately after invocation, even though the operation has not completed.
Primitives for communication (4)

Send and receive primitives: in a message passing communication, a message is sent and received by explicitly executing the send and receive primitives, respectively. The following are some definitions of primitives for communication:

Synchronous primitives: a send (or a receive) primitive is synchronous if both handshake with each other.

Asynchronous primitives: a send primitive is said to be asynchronous if control returns back to the invoking process after the data item to be sent has been copied out of the user-specified buffer. The asynchronous receive could be presented as a specific setting of the synchronous case (i.e no reply).
Primitives for communication (5)

Send and receive primitives: in a message passing communication, a message is sent and received by explicitly executing the send and receive primitives, respectively. The following are some definitions of primitives for communication:

There are therefore four versions of the send primitive (1) blocking synchronous (2) non-blocking synchronous (3) blocking asynchronous (4) non-blocking asynchronous. For the receive primitive, there are two versions (5) blocking synchronous (6) non-blocking asynchronous.
Primitives for communication (6)

The following are some definitions of primitives for communication:

(1) **Blocking synchronous send**: the data gets copied from the user buffer to the kernel buffer and is then sent over the network. After the data is copied to the receiver’s system buffer, an acknowledgement back to the sender causes control to return to the process and completes the send.

(5) **Blocking synchronous receive**: the receive call blocks until the data expected arrives and is written in the specified user buffer. Then control is returned to the user process.
The following are some definitions of primitives for communication:

(2) Non-blocking synchronous send: control returns back to the invoking process as soon as the copy of data from the user buffer to the kernel buffer is initiated. A parameter in the non-blocking call also gets set with the handle of a location. The user process can invoke the blocking wait operation on the returned handle.

(6) Non-blocking synchronous receive: the receive call will cause the kernel to register the call and return the handle of a location that the user process can later check for the completion. This location gets posted by the kernel after the expected data arrives and is copied to the user-specified buffer.

| Duration to copy data from or to user buffer |
| Duration in which the process issuing send or receive primitive is blocked |
| **S** | **Send** primitive issued | **S_C** | processing for **Send** completes |
| **R** | **Receive** primitive issued | **R_C** | processing for **Receive** completes |
| **P** | The completion of the previously initiated nonblocking operation |
| **W** | Process may issue **Wait** to check completion of nonblocking operation |
Primitives for communication (8)

The following are some definitions of primitives for communication:

(3) **Blocking asynchronous send**: the user process that invokes the send is blocked until the data is copied from the user’s buffer to the kernel buffer.

(4) **Non-blocking asynchronous send**: the user process that invokes the send is blocked until the transfer of the data from the user’s buffer to the kernel buffer is initiated. The asynchronous send completes when the data has been copied out of the user’s buffer.

---

| Duration to copy data from or to user buffer |
| Duration in which the process issuing send or receive primitive is blocked |
| $S$ | $Send$ primitive issued | $S_C$ | processing for $Send$ completes |
| $R$ | $Receive$ primitive issued | $R_C$ | processing for $Receive$ completes |
| $P$ | The completion of the previously initiated nonblocking operation |
| $W$ | Process may issue $Wait$ to check completion of nonblocking operation |
Introduction to distributed computing

1. Introduction to distributed systems
   1.1. Definitions, application domains and motivations
   1.2. Relations to parallel systems
   1.3. Goals, challenges and trends

2. Architecture, design, communication and computing
   2.1. The layered model and design issues
   2.2. Primitives for communication
   2.3. Distributed computing, definitions and challenges
Distributed computing, definitions and challenges

**Distributed computing:** is a field of computer science that studies distributed systems. It is also referred as the use of distributed systems to solve computational problems. A problem is divided into many tasks, each of which is solved by one or more computers. We briefly summarize some key algorithmic challenges in distributed computing.

**Time and global states in a distributed system:** the challenges is to provide accurate physical time, and a variant of time called logical time. Logical time is relative time, and eliminates the overheads of providing physical time for applications where physical time is not required. Due to the inherent distributed nature of the system, it is not possible for any one process to directly observe a meaningful global state across all the processes. Observing global states of a system involves a time dimension for consistent observation.

**Coordination and agreement:** the processes execute concurrently, except when they need to synchronize to exchange information. These are cooperating processes that are able to communicate with each other to work jointly in some activity. Such processes exhibit coordination and agreement. Coordination and agreement are essential for the distributed processes. Here are some examples of problems requiring coordination and agreement: group communication, leader election, mutual exclusion, deadlock detection and resolution, termination detection, garbage collection, etc.

**At the corner:** distributed program design and verification tools, debugging distributed programs, data replication, consistency models, and caching, distributed shared memory abstraction, reliable and fault-tolerant distributed systems, load balancing, etc.