Introduction to Distributed Systems

Corso di Sistemi Distribuiti e Cloud Computing
A.A. 2018/19

Valeria Cardellini

Technology advances

Networking

Computing power  Memory
Protocols  Storage
Internet evolution: 1977

![ARPANET Logical Map, March 1977](image)

Internet evolution: 2017

- IPv4 AS-level Internet graph
  - Geographic locations, number of customers, and interconnections of ~47000 Autonomous Systems (ASs)

Source: [www.caida.org/research/topology/as_core_network](www.caida.org/research/topology/as_core_network)
Internet growth: number of hosts

In 2014 it was the first time the survey measured a billion websites, a milestone achievement that was unimaginable two decades ago.

Source: Netcraft Web server survey

Web growth: number of Web servers
Metcalfe’s law

“The value of a telecommunications network is proportional to the square of the number of connected users of the system”.

Networking is **socially and economically** interesting

---

### Internet traffic in 2016

**Top 10 peak period applications (North America, mobile access)**

<table>
<thead>
<tr>
<th>Rank</th>
<th>Upstream</th>
<th>Downstream</th>
<th>Aggregate</th>
<th>Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Facebook</td>
<td>20.87%</td>
<td>YouTube</td>
<td>19.16%</td>
</tr>
<tr>
<td>2</td>
<td>SSL - OTHER</td>
<td>14.02%</td>
<td>Facebook</td>
<td>14.07%</td>
</tr>
<tr>
<td>3</td>
<td>Google Cloud</td>
<td>9.28%</td>
<td>HTTP - OTHER</td>
<td>9.36%</td>
</tr>
<tr>
<td>4</td>
<td>HTTP - OTHER</td>
<td>8.92%</td>
<td>SSL - OTHER</td>
<td>8.85%</td>
</tr>
<tr>
<td>5</td>
<td>YouTube</td>
<td>5.01%</td>
<td>Instagram</td>
<td>6.06%</td>
</tr>
<tr>
<td>6</td>
<td>Snapchat</td>
<td>4.36%</td>
<td>Snapchat</td>
<td>4.17%</td>
</tr>
<tr>
<td>7</td>
<td>Instagram</td>
<td>3.35%</td>
<td>Netflix</td>
<td>3.56%</td>
</tr>
<tr>
<td>8</td>
<td>BitTorrent</td>
<td>2.16%</td>
<td>iTunes</td>
<td>2.02%</td>
</tr>
<tr>
<td>9</td>
<td>FaceTime</td>
<td>1.97%</td>
<td>Google Cloud</td>
<td>2.87%</td>
</tr>
<tr>
<td>10</td>
<td>iCloud</td>
<td>1.82%</td>
<td>MPEG - OTHER</td>
<td>2.37%</td>
</tr>
</tbody>
</table>

**North American fixed access networks**
- Streaming audio and video: 71% of evening traffic
- Cloud storage: the largest source of upstream traffic

### Future Internet traffic


In 2016 **annual global IP traffic** was **1.2 ZB**; growing **3-fold** from 2016 to 2021 and will have increased **127-fold** from 2005 to 2021

- The **number of devices** connected to IP networks will be three times as high as the global population in 2021
- **Smartphone** traffic will **exceed** PC traffic by 2021. By 2021 PCs will account for only 25% and smartphones for 33% (46% and 13% in 2016)
- By 2021 traffic from **wireless and mobile devices** will account for more than **63%** of total IP; in 2014 only 46%
- By 2021 **Content Delivery Networks (CDNs)** will carry **71%** of all Internet video traffic; in 2014 only 45%
- In 2021 it would take an individual over **5 million years** to watch the amount of video that will cross global IP networks each month

Implication of this trend: Internet is replacing voice telephony, television... will be the dominant transport technology for everything

---

### Computing power

- Computers got…
  - Smaller
  - Cheaper
  - Power efficient
  - Faster

- **1974**: Intel 8080
  - 2 MHz, 6K transistors
- **2004**: Intel P4 Prescott
  - 3.6 GHz, 125 million transistors
- **2011**: Intel 10-core Xeon Westmere-EX
  - 3.33 GHz, 2.6 billion transistors
- **GPUs** scaled as well: in 2016 NVIDIA Pascal GPU
  - 60 streaming multiprocessors of 64 cores each, 150 billion transistors
  - Used for general-purpose computing (GPGPU)
Multicore processor and NVIDIA Pascal GPU

Overall architecture of NVIDIA Pascal GPU

Architecture of each streaming multiprocessor in NVIDIA Pascal GPU
Not only Internet and Web

- Internet and Web are two notable examples of distributed systems; others include:
  - Financial systems, HPC systems, server farms, clouds, … sometimes only accessible through Intranets
  - Peer-to-peer (P2P) systems
  - Home networks (home entertainment, multimedia sharing)
  - Wireless sensor networks
  - **Internet of Things (IoT)**

Gartner's annual IT hype cycle for emerging technologies
Where was cloud computing in 2014 and previous years?

Many of these technologies are strictly related to (and impossible without) distributed systems and Cloud computing!
Distributed system

- Multiple definitions of **distributed system (DS)**, not always coherent with each other
- [van Steen & Tanenbaum] A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system
  - Autonomous computing elements, also referred to as nodes, be they hardware devices or software processes
  - Users or applications perceive a single system: nodes need to collaborate

![Diagram of distributed system with middleware](image)

Distributed system

- [Coulouris & Dollimore] A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages
  - If components = CPUs we have the definition of MIMD (Multiple Instruction stream Multiple Data stream) parallel architecture

- [Lamport] A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable
  - Emphasis on fault tolerance
Who is Leslie Lamport?

• Recipient of 2013 Turing award (the Nobel for computer science)  
  https://bit.ly/2ONvnfA
• His research contributions have laid the foundations of the theory  
  and practice of distributed systems
• Fundamental concepts such as causality, logical clocks and  
  Byzantine failures; some notable papers:
  – “Time, Clocks, and the Ordering of Events in a Distributed System”
  – “The Byzantine Generals Problem”
  – “The Part-Time Parliament”
• Algorithms to solve many fundamental problems in distributed  
  systems, including:
  – Paxos algorithm for consensus
  – Bakery algorithm for mutual exclusion of multiple threads
  – Snapshot algorithm for consistent global states
• He is also the initial developer of LaTeX

Why to build distributed systems?

• Share resources
  – Resource = computing node, data, storage, network,  
    executable code, object, service, …
• Improve performance
• Improve dependability (availability, reliability, …)
• Bridge “geographical” distances
• Maintain autonomy
• Reduce costs
• Allow interaction
• Support Quality of Service (QoS)
• Improve security
Why to study distributed systems?

• Distributed systems are more complex than centralized ones
  – E.g., no global clock, group membership, …

• Building them is harder
  – … and building them correct is even much harder

• Managing, and, above all, testing them is difficult

Some distinguishing features of DS

• Concurrency
  – Centralized systems: a design choice
  – Distributed systems: a fact of life to be dealt with

• Absence of global clock
  – Centralized systems: use the computer’s physical clock for synchronization
  – Distributed systems: many clocks and not necessarily synchronized

• Independent and partial failures
  – Centralized systems: fail completely
  – Distributed systems: fail only partially (i.e., only a part of the DS), often due to communication; very difficult and in general impossible to hide partial failures and their recovery
Challenges in distributed systems

- Many challenges associated with designing distributed systems (and some of them are not new)
  - Heterogeneity
  - Distribution transparency
  - Openness
  - Scalability

While improving performance, system availability and reliability, guaranteeing security, energy efficiency, ...

Valeria Cardellini - SDCC 2018/19

Heterogeneity

- Levels:
  - Networks
  - Computer hardware
  - Operating systems
  - Programming languages
  - Multiple implementations by different developers

- The solution? **Middleware: the OS of DSs**

Middleware: software layer placed on top of OSs providing a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating systems and programming languages

Contains commonly used components and functions that need not be implemented by applications separately
Some middleware services

- Communication
- Transactions
- Service composition
- Reliability

Distribution transparency

- Distribution transparency: single coherent system where the distribution of processes and resources is transparent (i.e., invisible) to users and apps
- **Types** of distribution transparency (*ISO 10746*, Reference Model of Open Distributed Processing)

Access transparency
- Hide differences in data representation and how a resource is accessed
- E.g.: use the same mechanism for local or remote invocation

Location transparency
- Hide where the resource is located
  - E.g.: URL hides IP address
- Access + location transparency = network transparency

Migration transparency
- Hide that the resource may move to another location (even at runtime) without affecting operativeness
Distribution transparency (2)

Replication transparency
- Hide that there are multiple replicas of the same resource
  - Each replica should have the same name
  - Require also location transparency

Concurrency transparency
- Hide that the resource may be shared by several independent users
  - E.g.: concurrent access of multiple users to the same DB table
  - Concurrent access to shared resource should leave it in a consistent state; e.g., by using locking mechanisms

Failure transparency
- Hide the failure and recovery of a resource
- See the DS definition by Lamport

Degree of distribution transparency

- Aiming to full distribution transparency is often too much
  - Communication latencies cannot be always hidden: access from Rome to a resource located on a server in New York requires at least 23 ms
  - Impossible to completely hide failures in a large-scale DS
    - You cannot distinguish a slow computer from a failing one
    - You can never be sure that a server actually performed an operation before a crash
  - Full transparency costs in terms of performance
    - E.g.: keeping data replicas exactly up-to-date takes time
    - Tradeoff between degree of consistency and system performance (see slides on consistency in DSs)
Apertura

- Un SD aperto offre servizi che possono essere facilmente usati o integrati in altri sistemi
  - Il SD presenta interfacce standard e pubbliche
    - Scritte usando un IDL (Interface Definition Language)
    - Spesso l’interfaccia specifica la sintassi del servizio (nomi delle funzioni, parametri di I/O), ma non la semantica (cosa fa il servizio)
    - La definizione dell’interfaccia dovrebbe essere completa e neutrale
    - Alcuni esempi di IDL: Sun RPC, Thrift, WSDL, OMG IDL
  - Il SD supporta la portabilità delle applicazioni
    - Un’applicazione sviluppata per il SD A può essere eseguita, senza modifiche, sul SD B che espone le stesse interfacce di A
  - Il SD supporta l’interoperabilità
    - Implementazioni di sistemi (o componenti) di diversi produttori possono coesistere e collaborare basandosi unicamente sui reciproci servizi specificati da uno standard comune
  - Esempi: Java EE, .Net, Web Services

“Practice shows that many distributed systems are not as open as we’d like” (van Steen & Tanenbaum)

Separare politiche e meccanismi

- Per realizzare un SD aperto e flessibile occorre separare le politiche dai meccanismi
- Idealmente, il SD fornisce soltanto i meccanismi
- Es.: meccanismi in un Web browser
  - Supporto per il caching dei dati
- Es.: politiche in un Web browser
  - Quali risorse memorizzare in cache
  - Quanto a lungo memorizzare le risorse in cache
  - Quando effettuare il refresh delle risorse in cache
  - Cache privata o condivisa con altri utenti
- Risultato: molti parametri da configurare
- Possibile soluzione: sistema self-configurable

“Finding the right balance in separating policies from mechanisms is one of the reasons why designing a distributed system is sometimes more an art than a science” (van Steen & Tanenbaum)
Scalabilità

- Capacità di un sistema (distribuito) di mantenere un livello adeguato di prestazioni all’aumentare di:
  - risorse che lo compongono ed utenti ➔ scalabilità rispetto alla dimensione
  - distanza tra le risorse del SD e tra risorse del SD ed utenti ➔ scalabilità geografica
  - numero di domini amministrativi coinvolti ➔ scalabilità amministrativa

- La maggior parte dei SD si occupa della scalabilità rispetto alla dimensione
  - Due direzioni per la scalabilità rispetto alla dimensione
    - Verticale (scale-up): server più potente ovvero la classica non soluzione!
    - Orizzontale (scale-out)

Example: scaling with number of clients

- Google File System
  - Distributed file system that we’ll study later

- Scale parameter: number of clients
- Scalability metric: aggregated read (write, append) speed, assuming random file access
- Scalability criterion: the closer to network limit, the better
Tecniche per la scalabilità

• **Nascondere la latenza nella comunicazione**
  Non attendere senza far niente la risposta di un servizio remoto, ma nel frattempo far fare altro lavoro utile al richiedente
  – Come? Usando la **comunicazione asincrona**
    • Handler (gestore) specifico per completare la richiesta in arrivo
  – **Problema**: non adatta per tutte le tipologie di applicazioni (ad es. applicazioni molto interattive)

• **Suddividere e distribuire computazione e dati**
  Suddividere computazione e dati in parti più piccole e distribuirli tra molteplici nodi del SD
  – Es.: servizi di naming decentralizzati (DNS), approcci per la computazione distribuita (MapReduce)

• **Replicare componenti del SD e dati**
  Rendere disponibili repliche dei componenti del SD e dei dati su molteplici nodi del SD
  – Es.: Dropbox (servizio di Cloud storage) in cui i dati sono memorizzati localmente sul PC dell’utente e su molteplici server distribuiti gestiti da Dropbox

Problemi per la scalabilità

• A prima vista sembra facile applicare le tecniche per la scalabilità, ma…
  – Molteplici repliche ➔ **problemi di consistenza**
    • La replica modificata diviene diversa dalle altre repliche
  – Per mantenere le repliche consistenti tra loro occorre una sincronizzazione globale di tutte le repliche ad ogni modifica
  – Ma la la sincronizzazione globale preclude soluzioni scalabili su larga scala!
    • Es.: la rete può subire un partizionamento

• Tuttavia, se è possibile **tollerare un certo grado di inconsistenza**, si può ridurre il bisogno di sincronizzazione globale
  – Il grado di inconsistenza tollerabile **dipende dal tipo di applicazione**
    • Esempi: blog, scambi di borsa, aste on-line, controllo del traffico aereo, …
Fallacies in realizing distributed systems

- Many distributed systems are needlessly complex because of errors in design and implementation that were patched later
  1. The network is reliable
     - "You have to design distributed systems with the expectation of failure" (Ken Arnold)
  2. Latency is zero
     - Latency is more problematic than bandwidth
     - "At roughly 300,000 kilometers per second, it will always take at least 30 milliseconds to send a ping from Europe to the US and back, even if the processing would be done in real time." (Ingo Rammer)
  3. Bandwidth is infinite
  4. The network is secure
  5. Topology does not change
     - That's right, it doesn't--as long as it stays in the test lab!
  6. There is one administrator
  7. Transport cost is zero
     - Going from the application level to the transport level is not free
     - The costs for setting and running the network are not free
  8. The network environment is homogeneous

Do not think that technology solves everything!

See Fallacies of Distributed Computing Explained
Three types of distributed systems

• High performance distributed computing systems
  – Cluster computing
  – Grid computing
  – Cloud computing
    • We'll focus on it later

• Distributed information systems

• Distributed pervasive systems

Cluster computing

• Cluster: insieme di nodi di computazione ad alte prestazioni interconnessi tramite una rete locale ad alta velocità
  – Omogeneità: nodi con stesso sistema operativo, hardware molto simile, connessione attraverso la stessa rete

• Obiettivi di alte prestazioni (HPC o High Performance Computing) e/o elevata affidabilità (HA o High Availability)

• Organizzazione gerarchica con singolo nodo principale (ad es. Beowulf) oppure Single System Image (ad es. MOSIX)
  – Cosa è MOSIX: sistema operativo per cluster eseguibile su Linux nativo o macchina virtuale
  – Cosa offre MOSIX: bilanciamento automatico del carico, scoperta automatica delle risorse, migrazione di processi, …
A typical cluster architecture

- A cluster of servers interconnected by a high-bandwidth SAN or LAN with shared I/O devices and disk arrays

Clusters dominate Top 500 architectures

- See the architecture share of Top 500 systems [www.top500.org](http://www.top500.org)
- Clusters in June 2018
  - Systems share = 87.4%
  - Performance share = 65.14%
- MPP in June 2018
  - MPP: Massively Parallel Processing
  - Systems share = 12.6%
  - Performance share = 34.86%
Grid computing

• The next step of cluster computing

• Grid: large, widely distributed and heterogeneous clusters interconnected by high-speed network links over selected resource sites
  – Heterogeneity: nodes (=clusters) may differ in hardware, software, networks, security policies, …
  – Nodes may be located in different administrative domains

• Computational Grid
  – Provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities

• Data Grid
  – Deals with the controlled sharing and management of large amounts of distributed data

• Examples
  – TeraGrid, GriPhyN, UK EGEE, D-Grid, ChinaGrid

Distributed information systems

• Among distributed information systems let us consider transaction processing systems

BEGIN_TRANSACTION(server, transaction);
READ(transaction, file1, data);
WRITE(transaction, file2, data);
newData := MODIFIED(data);
IF WRONG(newData) THEN
  ABORT_TRANSACTION(transaction);
ELSE
  WRITE(transaction, file2, newData);
  END_TRANSACTION(transaction);
END IF;

– The effect of all READ and WRITE operations become permanent only with END_TRANSACTION
– A transaction is an atomic operation ("all-or-nothing")
Transazione

- Transazione: insieme di operazioni sullo stato di un oggetto che soddisfa le proprietà **ACID**
  - **Atomicità**
    - La transazione o viene eseguita completamente (come un’azione singola, indivisibile ed istantanea) o non viene eseguita affatto
  - **Consistenza**
    - La transazione non viola le invarianti del sistema
  - **Isolamento**
    - Transazioni concorrenti non interferiscono le une con le altre
  - **Durabilità**
    - Una volta che la transazione ha reso effettive le modifiche, esse sono permanenti

Distributed transactions

- **Distributed** (or nested) transaction: composed by multiple sub-transactions which are distributed across several servers
  - *Transaction Processing (TP) Monitor*: responsible for coordinating the execution of the distributed transaction
  - Example: Oracle Tuxedo

- We’ll study distributed commit protocols
Integrating applications

• Organizations confronted with many networked applications, how to achieve interoperability?

• Simple integration
  – Clients combine requests for different applications; send that off; collect responses, and present a coherent result to the user
  – Can work but…

• Enterprise Application Integration (EAI)
  – Integration framework composed of a collection of technologies and services which form a middleware to enable integration of systems and applications across the enterprise

• Middeware and EAI
  – Middleware offers communication facilities for integration

Integrating applications

• Communication among heterogeneous applications (communication middleware)
  – Remote Procedure Call (RPC)
  – Remote Method Invocation (RMI)
  – Message Oriented Middleware (MOM)
  – Enterprise Service Bus (ESB)
Distributed pervasive systems

- Distributed systems whose nodes are often
  - small, mobile, battery-powered and often embedded in a larger system
  - characterized by the fact that the system naturally blends into the user's environment

- Three (overlapping) subtypes of pervasive systems
  - Ubiquitous computing systems: pervasive and continuously present, i.e. there is a continuous interaction between system and users
  - Mobile computing systems: pervasive, with emphasis on the fact that devices are inherently mobile
  - Sensor networks: pervasive, with emphasis on the actual (collaborative) sensing and actuation of the environment

Ubiquitous computing systems

- Basic characteristics
  - Distribution: devices are networked, distributed, and accessible in a transparent manner
  - Interaction: interaction between users and devices is highly unobtrusive
  - Context awareness: the system is aware of a user's context (location, identity, time, activity) in order to optimize interaction
  - Autonomy: devices operate autonomously without human intervention, and are thus highly self-managed
  - Intelligence: the system as a whole can handle a wide range of dynamic actions and interactions
Mobile computing systems

• Mobile computing systems are generally a subclass of ubiquitous computing systems
  – Meet all of the five basic characteristics

• Typical characteristics
  – Many different types of mobile devices: smart phones, remote controls, car equipment, …
  – Wireless communication
  – Devices may continuously change their location
    • Setting up a route may be problematic, as routes can change frequently
    • Devices may easily be temporarily disconnected, e.g., disruption-tolerant networks in MANETs
    • We’ll study flooding and gossiping techniques to spread messages

Sensor networks

• Sensors
  – Many (10-10^6)
  – Simple: limited computing, memory and communication capacity
  – Often battery-powered (or even battery-less)
  – Failures are frequent

• Sensor networks as distributed systems
  (a) Store and process data in a centralized way only on the sink node
  (b) Store and process data in a distributed way on the sensors (active and autonomous)
Examples of wireless sensor networks (WSNs)

Agricultural WSNs

Underwater WSNs