Chapter 1

Introduction to Distributed Systems

September, 17th

Introduction

Introduction

What’s a distributed system?

Introduction

What's a distributed system?

- "A distributed system is a *collection* of *autonomous* computing elements that appears to its users as a *single coherent* system."
  — A. S. Tanenbaum and M. van Steen

  Focus on **transparency**

What's a distributed system?

- "A distributed system is a *collection* of *autonomous* computing elements that appears to its users as a *single coherent* system."
  — A. S. Tanenbaum and M. van Steen

  Focus on **transparency**

- "A system in which hardware and software components located at *networked* computers communicate and coordinate their actions only by *passing messages*."
  — Coulouris et al.

  "A system is distributed if the message transmission delay is not negligible compared to the time between events in a single process."
  — Leslie Lamport

  Focus on **communication**

From early computers to distributed systems

What makes distributed systems possible?

- A revolution that started in the mid 80’s
- Powerful microprocessors
  - 64-bit CPUs, multi-core CPUs, …
- High-speed computer networks
  - Local-area networks (low latency, high bandwidth)
  - Wide-area networks (high bandwidth)

From early computers to distributed systems

What makes distributed systems possible?

- A revolution that started in the mid 80’s
- Powerful microprocessors
    - 64-bit CPUs, multi-core CPUs, …
- High-speed computer networks
    - Local-area networks (low latency, high bandwidth)
    - Wide-area networks (high bandwidth)

What makes distributed systems desirable?

From early computers to distributed systems

What makes distributed systems possible?

- A revolution that started in the mid 80’s
- Powerful microprocessors
  - 64-bit CPUs, multi-core CPUs, …
- High-speed computer networks
  - Local-area networks (low latency, high bandwidth)
  - Wide-area networks (high bandwidth)

What makes distributed systems desirable?

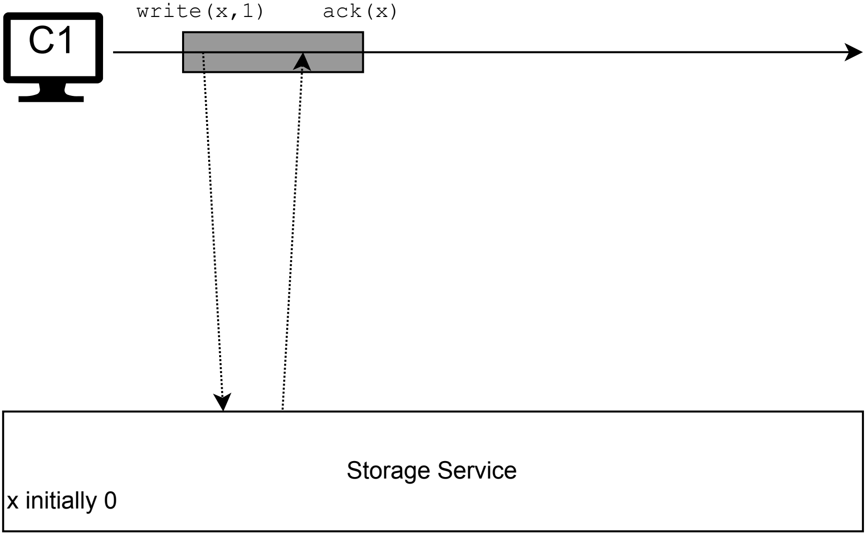
- Performance
- Cost
- Reliability / Availability
- Distributed information

Introduction

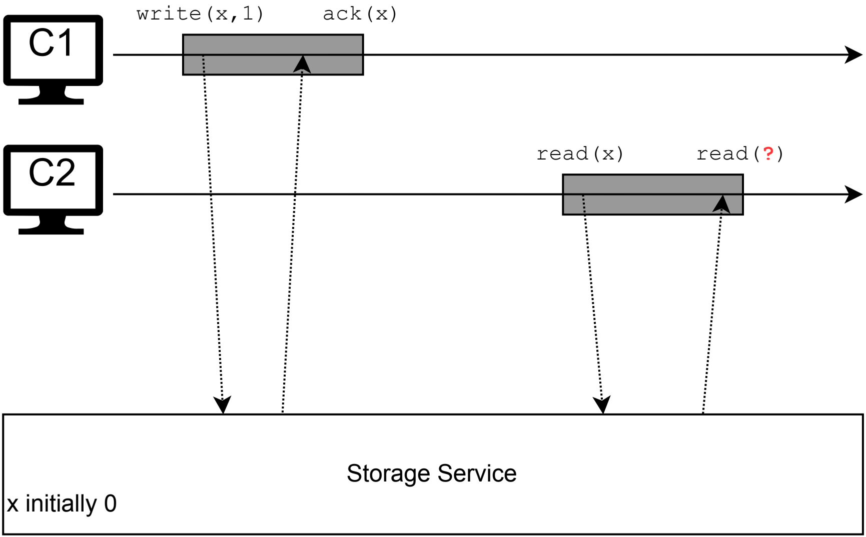
Significant consequences of distributed systems:

- Concurrency
- Absence of global clock
- Partial failures

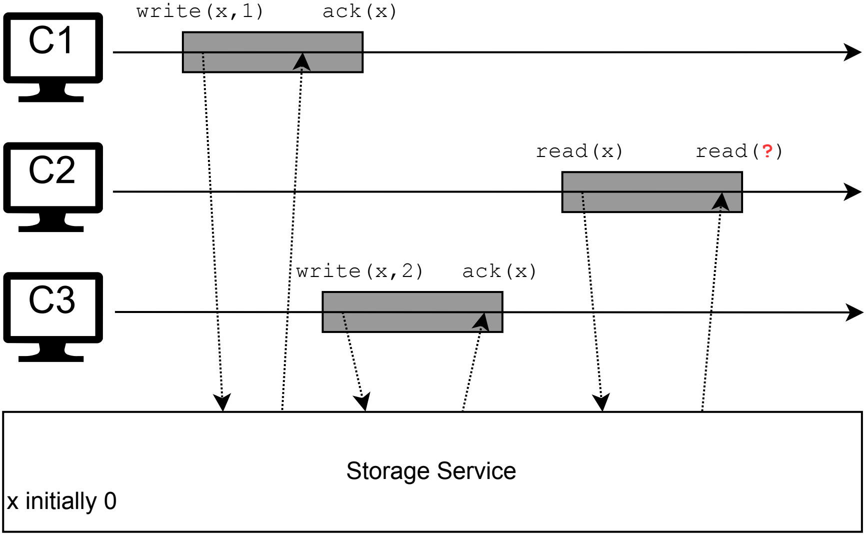
Concurrency



Concurrency

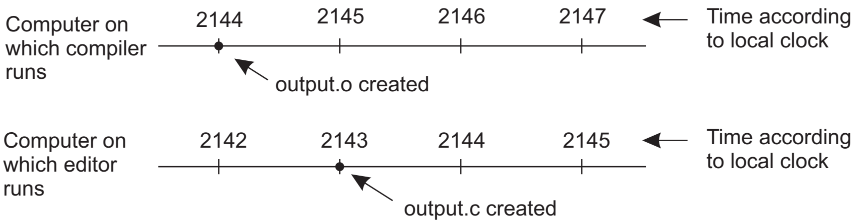


Concurrency



Absence of global clock

Example: Unix `make` program



When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Partial failures

```
Received: by jumbo.dec.com (5.54.3/4.7.34)
    id AA09105; Thu, 28 May 87 12:23:29 PDT
Date: Thu, 28 May 87 12:23:29 PDT
From: lamport ( Leslie Lamport )
To: src-t
Subject: distribution

There has been considerable debate over the years about what
constitutes a distributed system. It would appear that the following
definition has been adopted at SRC:

    A distributed system is one in which the failure of a computer
    you didn't even know existed can render your own computer
    unusable.

The current electrical problem in the machine room is not the
culprit--it just highlights a situation that has been getting
progressively worse. It seems that each new version of the nub makes
my FF more dependent upon programs that run elsewhere.

[…]

Leslie
```

Examples of distributed systems

Finance and commerce	eCommerce e.g. Amazon and eBay, PayPal, online banking and trading
The information society	Web information and search engines, ebooks, Wikipedia, social networking: Facebook and MySpace
Creative industries and entertainment	online gaming, music and film in the home, user-generated content, e.g., YouTube, Flickr
Healthcare	health informatics, online patient records, monitoring patients
Education	e-learning, virtual learning environments; distance learning
Transport and logistics	GPS in route finding systems, map services: Google Maps, Google Earth
Science	The Grid as an enabling technology for collaboration between scientists
Environmental management	sensor technology to monitor earthquakes, floods or tsunamis

from DISTRIBUTED SYSTEMS Concepts and Design, 5th ed G. Coulouris et al., 2012

Design Goals

Design Goals

- **Openness**: easy to integrate with other systems
- **Scalability**: easy to make “bigger”
- **Transparency**: easy to use (ignore distribution)

Openness

- Offering services according to standard rules
- Describe both *syntax* and *semantics* of services
- Typical in computer networks
  - RFCs: RFC 9112 (HTTP syntax) and RFC 9110 (HTTP semantics)
- Interface Definition Language (IDL). E.g., gRPC:

```
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}
message HelloRequest {
  string name = 1;
}
message HelloReply {
  string message = 1;
}
```

Scalability

Dimensions of scalability:

- **System size** (e.g., adding more users)
  Problem: insufficient resources (CPU, Network, IO)
- **Geography** (e.g., users may lie far apart)
  Problem: high network latency ($\approx 3\mu s \ / \ km$)
- **Administration** (e.g., multiple organizations)
  Problem: conflicting policies (security, resource, management)

Scalability

Dimensions of scalability:

- **System size** (e.g., adding more users)
  Problem: insufficient resources (CPU, Network, IO)
- **Geography** (e.g., users may lie far apart)
  Problem: high network latency ($\approx 3\mu s\ /\ km$)
- **Administration** (e.g., multiple organizations)
  Problem: conflicting policies (security, resource, management)

Factors limiting scalability:

Concept	**Example**
Centralized deployment	A single server for all users
Centralized data	A single online telephone book
Centralized algorithms	Doing routing based on complete information

Examples of scalability bottlenecks

Scaling techniques

Hiding communication latencies

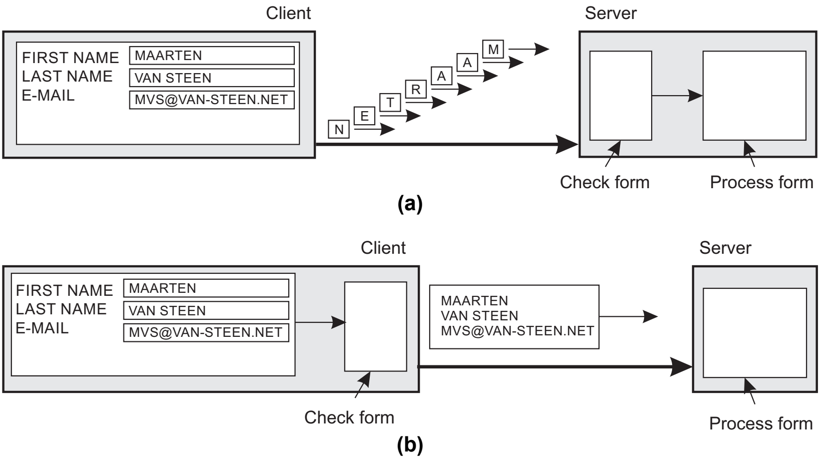
- Important for geographical scalability
- Avoid waiting for responses to remote service requests
  - asynchronous communication (as opposed to RPC-like)
    e.g., batch processing and parallel applications

Scaling techniques

Hiding communication latencies

- Important for geographical scalability
- Avoid waiting for responses to remote service requests
  - asynchronous communication (as opposed to RPC-like)
    e.g., batch processing and parallel applications
- What about interactive applications?
  - reduce communication if possible
    e.g., validate database forms at the application side

Scaling techniques



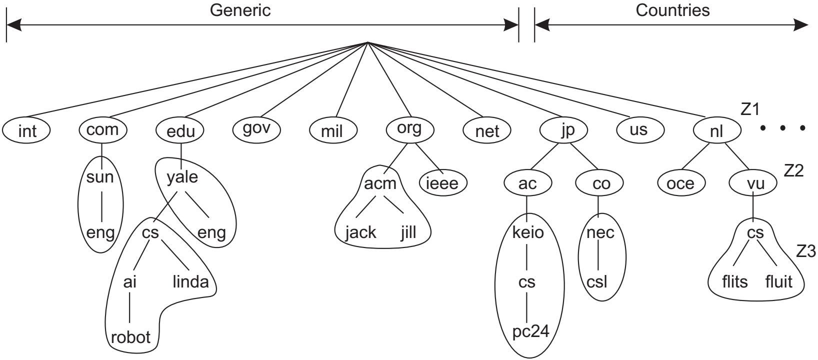
The difference between letting (a) a server or (b) a client check forms as they are being filled.

Scaling techniques

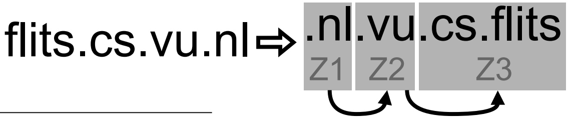
Partitioning and distribution

- *Partition* component into parts, *distribute* parts across the system
- e.g., World Wide Web (WWW)
- e.g., Domain Name System (DNS)
  - Tree of domains, divided into non-overlapping zones
  - Name in a zone handled by a single name server

Scaling techniques



An example of dividing the DNS name space into zones[1].



[1] one can play with `dig -t NS ..`

Scaling techniques

Replication

- To increase performance (e.g., load balancing)
- To reduce latency (e.g., reading from a nearby replica in WANs)
- To increase availability (e.g., tolerate node crashes)
- Caching
    - Special form of replication, typically done on demand
- **But!** How to keep replicas consistent?
    - Strong versus weak consistency

Transparency

Transparency	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

Different forms of transparency in a distributed system (ISO, 1995)

Transparency

- Access transparency
  - Hide difference in data representation
  - Hide difference in machine architectures
  - e.g., unify different file-naming conventions in different OSs
- Location transparency
  - Users cannot tell where a resource is physically located
  - Key mechanism: assign logical names to resources
    - e.g., URL's
  - Need of a naming service (logical name $\mapsto$ physical address)
- Relocation transparency
  - *System* moving resources without user noticing
- Migration transparency
  - Support for mobility of *user* processes and resources

Transparency

- Replication transparency
    - Hiding the fact that multiple replicas of a resource exist
    - Replication is a key mechanism to increase *availability* and *performance*
- Concurrency transparency
    - How to handle multiple users accessing the same resource?
    - Transactions is one possible abstraction
- Failure transparency
    - Users do not notice when a resource fails to work properly

Design pitfalls

Remember: components are dispersed across a *network*

False assumptions:

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator

Types of Distributed Systems

Types of Distributed Systems

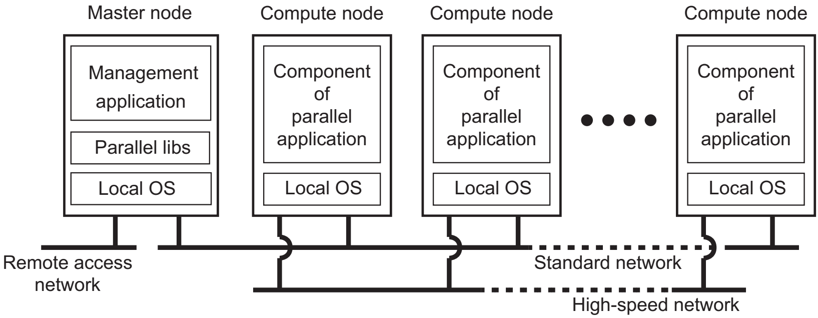
- Distributed computing systems
- Distributed information systems
- Pervasive (ubiquitous) systems

Distributed computing systems

Aim: High-performance computing tasks

- Cluster computing
  - Local-area network, same OS
  - Common administrative domain
- Grid computing
  - Federation of computer systems
  - Different administrative domains, hardware, software, ...
- Cloud computing
  - Third-party resources
  - Dynamically construct infrastructure
  - Goes beyond high-performance computing
    - e.g., ready-to-use storage and communication services

Cluster computing



An example of a cluster computing system.

Distributed computing systems

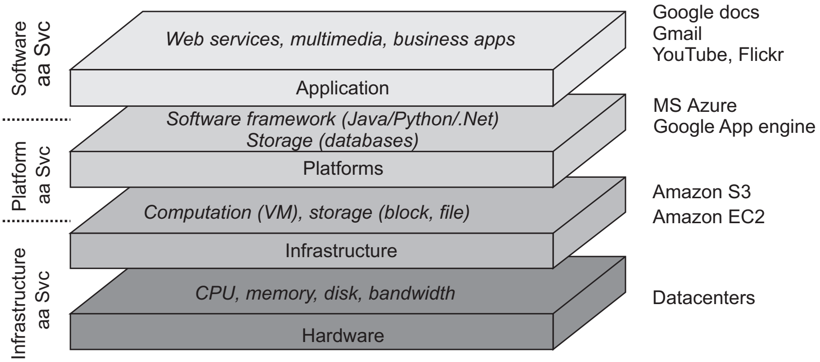
Grid computing

- *Resources* from *different organizations* brought together to allow *collaboration* of a group of people or organizations
- Focus on architectural issues: provide access to resources (servers, storage, databases, ...) from different administrative domains

Distributed computing systems

Cloud computing

- Basis: easily accessible pool of virtualized resources
- Utility-based computing, i.e., pay per-resource
- Dynamic resource allocation ⇒ easy to scale resources



The organization of clouds

Transaction processing systems

- Database applications and transactions

Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

Example primitives for transactions.

Transaction processing systems

- Database applications and transactions

Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

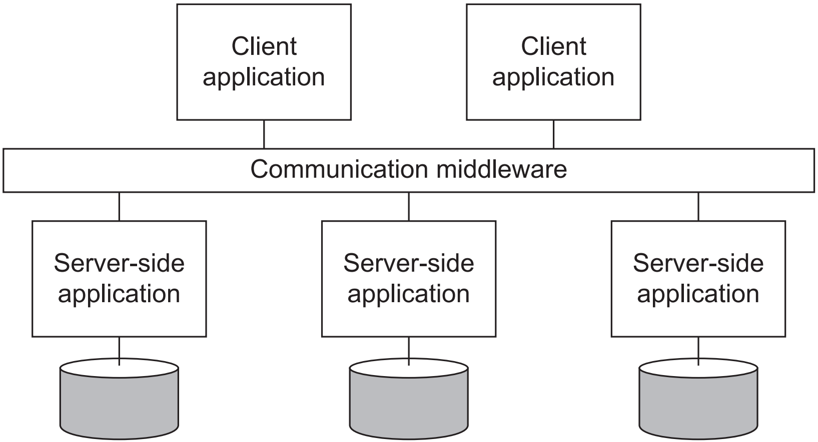
Example primitives for transactions.

- Properties of transactions (ACID)
  - *Atomic*: To the outside world, the transaction happens indivisibly
  - *Consistent*: The transaction does not violate system invariants
  - *Isolated*: Concurrent transactions do not interfere with each other
  - *Durable*: Once a transaction commits, the changes are permanent

Enterprise application integration

- Departure from the request/reply transaction processing model
- Application components communicate directly with each other
- Several types of communication exist
    - Remote procedure call (RPC)
    - Remote method invocation (RMI)
    - Publish/subscribe systems (pub-sub)

Enterprise application integration



Middleware as a communication facilitator in enterprise application integration.

Distributed pervasive systems

aka *Ubiquitous* computing systems

- **Computing elements:** mobile and embedded devices
  - small
  - battery-powered
  - mobile
  - wireless connection
  - heterogeneous

Distributed pervasive systems

aka *Ubiquitous* computing systems

- **Computing elements:** mobile and embedded devices
  - small
  - battery-powered
  - mobile
  - wireless connection
  - heterogeneous
- Requirements:
  - **distribution:** openness, scalability, transparency
  - ***interaction***: user and components interact unobtrusively
  - ***context-awareness***: context is an input to any interaction
  - ***autonomy***: components operate without user intervention
  - ***intelligence***: handles wide range of actions and interactions

Distributed pervasive systems

Ubiquitous computing

Example: Smart Home System

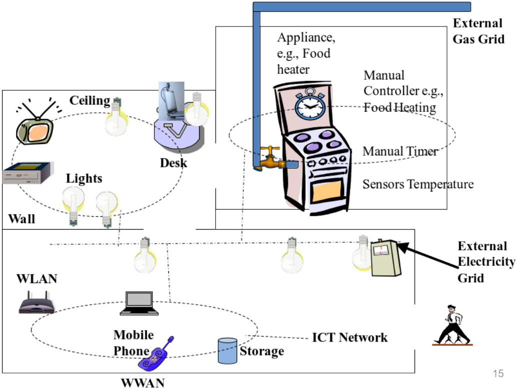


Illustration of a smart home system from *Ubiquitous Computing: Smart Devices, Environments and Interactions (S. Poslad)*

Requirements:

- distribution
- **interaction**
- **context-awareness**
- **autonomy**
- **intelligence**

Distributed pervasive systems

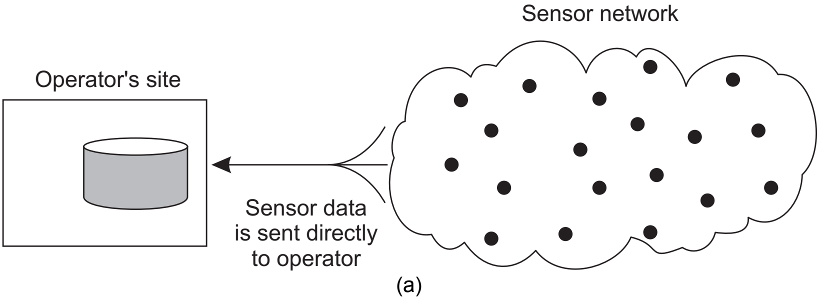
Sensor networks

- 10s to 100s to 1000s of small nodes, equipped with sensors
- Wireless communication, battery-powered nodes
- Similar to a network of distributed databases

Distributed pervasive systems

Sensor networks

- 10s to 100s to 1000s of small nodes, equipped with sensors
- Wireless communication, battery-powered nodes
- Similar to a network of distributed databases

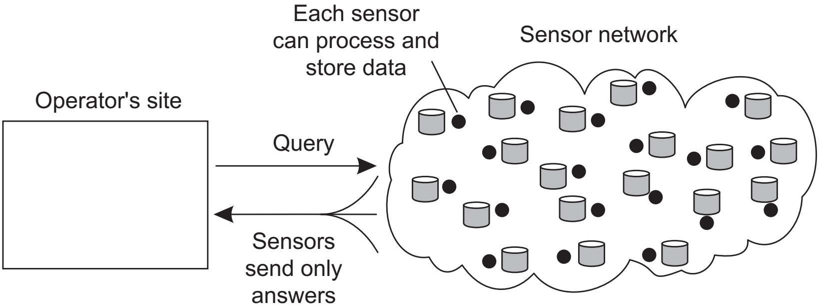


Organizing a sensor network database, while storing and processing data (a) only at the operator's site or...

Distributed pervasive systems

Sensor networks

- 10s to 100s to 1000s of small nodes, equipped with sensors
- Wireless communication, battery-powered nodes
- Similar to a network of distributed databases



(b)

Organizing a sensor network database, while storing and processing data... or (b) only at the sensors.