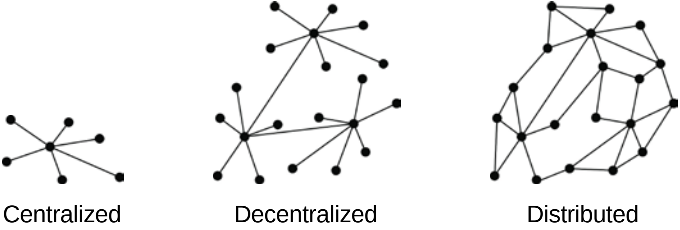
Chapter 1
Introduction

Distributed Systems

Distributed versus Decentralized

What many people state



Distributed versus Decentralized

What many people state

Centralized Decentralized Distributed

When does a decentralized system become distributed?

Distributed versus Decentralized

What many people state

Centralized Decentralized Distributed

When does a decentralized system become distributed?

- Adding 1 link between two nodes in a decentralized system?

Distributed versus Decentralized

What many people state

Centralized Decentralized Distributed

When does a decentralized system become distributed?

- Adding 1 link between two nodes in a decentralized system?
- Adding 2 links between two other nodes?

Distributed versus Decentralized

What many people state

Centralized Decentralized Distributed

When does a decentralized system become distributed?

- Adding 1 link between two nodes in a decentralized system?
- Adding 2 links between two other nodes?
- In general: adding $k > 0$ links....?

Alternative approach

Two views on realizing distributed systems

- Integrative view: connecting existing networked computer systems into a larger a system.
- Expansive view: an existing networked computer systems is extended with additional computers

Alternative approach

Two views on realizing distributed systems

- Integrative view: connecting existing networked computer systems into a larger a system.
- Expansive view: an existing networked computer systems is extended with additional computers

Two definitions

- A decentralized system is a networked computer system in which processes and resources are necessarily spread across multiple computers.
- A distributed system is a networked computer system in which processes and resources are sufficiently spread across multiple computers.

What are Distributed Systems?

A group of separate computers working together to achieve a certain shared goal.

These computers, also known as nodes, collaborate, exchange information via a network, and plan their actions to accomplish a shared objective by sharing resources, tasks, and data.

Some common misconceptions

Centralized solutions do not scale

Make distinction between logically and physically centralized. The root of the Domain Name System:

- logically centralized
- physically (massively) distributed
- decentralized across several organizations

Centralized solutions have a single point of failure

Generally not true (e.g., the root of DNS). A single point of failure is often:

- easier to manage
- easier to make more robust

Perspectives on distributed systems

Distributed systems are complex: take persepctives

- Architecture: common organizations
- Process: what kind of processes, and their relationships
- Communication: facilities for exchanging data
- Coordination: application-independent algorithms
- Naming: how do you identify resources?
- Consistency and replication: performance requires of data, which need to be the same
- Fault tolerance: keep running in the presence of partial failures
- Security: ensure authorized access to resources

What do we want to achieve?

Overall design goals

- Support sharing of resources
- Distribution transparency
- Openness
- Scalability

Sharing resources

Canonical examples

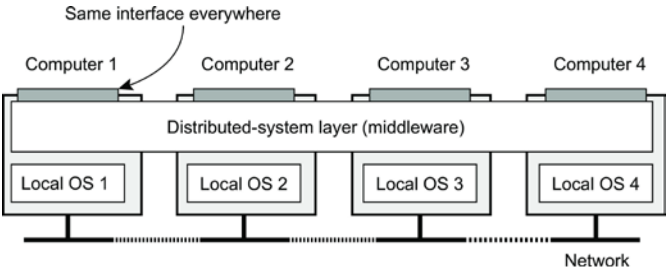
- Cloud-based shared storage and files
- Peer-to-peer assisted multimedia streaming
- Shared mail services (think of outsourced mail systems)
- Shared Web hosting (think of content distribution networks)

Observation

“The network is the computer”

(quote from John Gage, then at Sun Microsystems)

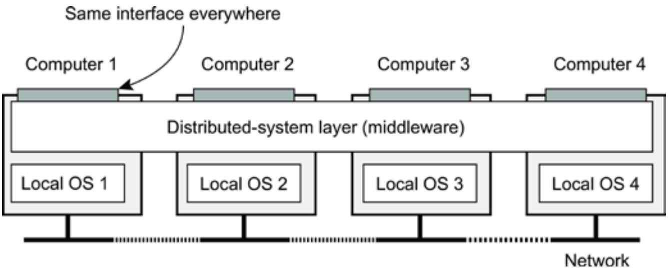
Distribution transparency



What is transparency?

The phenomenon by which a distributed system attempts to hide the fact that its processes and resources are physically distributed across multiple computers, possibly separated by large distances.

Distribution transparency



What is transparency?

The phenomenon by which a distributed system attempts to hide the fact that its processes and resources are physically distributed across multiple computers, possibly separated by large distances.

Observation

Distribution transparancy is handled through many different techniques in a layer between applications and operating systems: a middleware layer

Distribution transparency

Types

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

Degree of transparency

Aiming at full distribution transparency may be too much

Degree of transparency

Aiming at full distribution transparency may be too much

- There are communication latencies that cannot be hidden

Degree of transparency

Aiming at full distribution transparency may be too much

- There are communication latencies that cannot be hidden
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - 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

Degree of transparency

Aiming at full distribution transparency may be too much

- There are communication latencies that cannot be hidden
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - 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 will cost performance, exposing distribution of the system
  - Keeping replicas exactly up-to-date with the master takes time
  - Immediately flushing write operations to disk for fault tolerance

Degree of transparency

Exposing distribution may be good

- Making use of location-based services (finding your nearby friends)
- When dealing with users in different time zones
- When it makes it easier for a user to understand what's going on (when e.g., a server does not respond for a long time, report it as failing).

Degree of transparency

Exposing distribution may be good

- Making use of location-based services (finding your nearby friends)
- When dealing with users in different time zones
- When it makes it easier for a user to understand what's going on (when e.g., a server does not respond for a long time, report it as failing).

Conclusion

Distribution transparency is a nice goal, but achieving it is a different story, and it should often not even be aimed at.

Openness of distributed systems

Open distributed system

A system that offers components that can easily be used by, or integrated into other systems. An open distributed system itself will often consist of components that originate from elsewhere.

What are we talking about?

Be able to interact with services from other open systems, irrespective of the underlying environment:

- Systems should conform to well-defined interfaces
- Systems should easily interoperate
- Systems should support portability of applications
- Systems should be easily extensible

Policies versus mechanisms

Implementing openness: policies

- What level of consistency do we require for client-cached data?
- Which operations do we allow downloaded code to perform?
- Which QoS requirements do we adjust in the face of varying bandwidth?
- What level of secrecy do we require for communication?

Implementing openness: mechanisms

- Allow (dynamic) setting of caching policies
- Support different levels of trust for mobile code
- Provide adjustable QoS parameters per data stream
- Offer different encryption algorithms

On strict separation

Observation

The stricter the separation between policy and mechanism, the more we need to ensure proper mechanisms, potentially leading to many configuration parameters and complex management.

Finding a balance

Hard-coding policies often simplifies management, and reduces complexity at the price of less flexibility. There is no obvious solution.

Dependability

Basics

A component provides services to clients. To provide services, the component may require the services from other components ⇒ a component may depend on some other component.

Specifically

A component C depends on C^* if the correctness of C's behavior depends on the correctness of C^*'s behavior. (Components are processes or channels.)

Dependability

Requirements related to dependability

Requirement	Description
Availability	Readiness for usage
Reliability	Continuity of service delivery
Safety	Very low probability of catastrophes
Maintainability	How easy can a failed system be repaired

Reliability versus availability

Reliability $R(t)$ of component C

Conditional probability that C has been functioning correctly during $[0,t)$ given C was functioning correctly at the time $T = 0$.

Traditional metrics

- Mean Time To Failure (*MTTF*): The average time until a component fails.
- Mean Time To Repair (*MTTR*): The average time needed to repair a component.
- Mean Time Between Failures (*MTBF*): Simply *MTTF* + *MTTR*.

Terminology

Failure, error, fault

Term	Description	Example
Failure	A component is not living up to its specifications	Crashed program
Error	Part of a component that can lead to a failure	Programming bug
Fault	Cause of an error	Sloppy programmer

On security

Observation

A distributed system that is not secure, is not dependable

On security

Observation

A distributed system that is not secure, is not dependable

What we need

- Confidentiality: information is disclosed only to authorized parties
- Integrity: Ensure that alterations to assets of a system can be made only in an authorized way

On security

Observation

A distributed system that is not secure, is not dependable

What we need

- Confidentiality: information is disclosed only to authorized parties
- Integrity: Ensure that alterations to assets of a system can be made only in an authorized way

Authorization, Authentication, Trust

- Authentication: verifying the correctness of a claimed identity
- Authorization: does an identified entity has proper access rights?
- Trust: one entity can be assured that another will perform particular actions according to a specific expectation

Security mechanisms

Keeping it simple

It's all about encrypting and decrypting data using security keys.

Notation

$K(data)$ denotes that we use key K to encrypt/decrypt $data$.

Security mechanisms

Symmetric cryptosystem

With encryption key $E_K(data)$ and decryption key $D_K(data)$:
if $data = D_K(E_K(data))$ *then* $D_K = E_K$. Note: encryption and descryption key are the same and should be kept secret.

Asymmetric cryptosystem

Distinguish a public key $PK(data)$ and a private (secret) key $SK(data)$.

- Encrypt message from *Alice* to *Bob*: $data = \underbrace{SK_{bob}(\overbrace{PK_{bob}(data)}^{\text{Sent by Alice}})}_{\text{Action by Bob}}$
- Sign message for *Bob* by *Alice*:
  $[data, \underbrace{data \stackrel{?}{=} PK_{alice}(SK_{alice}(data))}_{\text{Check by Bob}}] = \underbrace{[data, SK_{alice}(data)]}_{\text{Sent by Alice}}$

Security mechanisms

Secure hashing

In practice, we use secure hash functions: $H(data)$ returns a fixed-length string.

- Any change from *data* to $data^*$ will lead to a completely different string $H(data^*)$.
- Given a hash value, it is computationally impossible to find a *data* with $h = H(data)$

Security mechanisms

Secure hashing

In practice, we use secure hash functions: $H(data)$ returns a fixed-length string.

- Any change from *data* to $data^*$ will lead to a completely different string $H(data^*)$.
- Given a hash value, it is computationally impossible to find a *data* with $h = H(data)$

Practical digital signatures

Sign message for *Bob* by *Alice*:

$$[data, \underbrace{H(data) \stackrel{?}{=} PK_{alice}(sgn)}_{\text{Check by Bob}}] = \underbrace{[data, H, sgn = SK_{alice}(H(data))]}_{\text{Sent by Alice}}$$

Scale in distributed systems

Observation

Many developers of modern distributed systems easily use the adjective "scalable" without making clear why their system actually scales.

Scale in distributed systems

Observation

Many developers of modern distributed systems easily use the adjective "scalable" without making clear why their system actually scales.

At least three components

- Number of users or processes (size scalability)
- Maximum distance between nodes (geographical scalability)
- Number of administrative domains (administrative scalability)

Scale in distributed systems

Observation

Many developers of modern distributed systems easily use the adjective "scalable" without making clear why their system actually scales.

At least three components

- Number of users or processes (size scalability)
- Maximum distance between nodes (geographical scalability)
- Number of administrative domains (administrative scalability)

Observation

Most systems account only, to a certain extent, for size scalability. Often a solution: multiple powerful servers operating independently in parallel. Today, the challenge still lies in geographical and administrative scalability.

Advantages of Distributed Systems

- Improved performance and scalability
- Increased fault tolerance and Reliability
- Enhanced reliability
- Enhanced Data Sharing
- Geographic flexibility
- Cost-effectiveness

Disadvantages of Distributed Systems

- Complexity
- Increased cost of set-up
- Security vulnerabilities
- Data inconsistency
- Network and resource overload
- Difficulty in debugging and troubleshooting

Application Areas of Distributed Systems

Distributed Systems are Every Where These days!

- Peer to Peer Networks
- E-Commerce
- Cloud Computing
- The Internet and Web Wide Web
- Online Gaming
- Video Conferencing Platforms
- Aircraft and Industrial Control Systems

END