Distributed Systems

(3rd Edition)

Chapter 01: Introduction

Version: March 20, 2022

Distributed System

Definition

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

Characteristic features

- Autonomous computing elements, also referred to as nodes, be they hardware devices or software processes.
- Single coherent system: users or applications perceive a single system ⇒ nodes need to collaborate.

Collection of autonomous nodes

Independent behavior

Each node is autonomous and will thus have its own notion of time: there is no global clock. Leads to fundamental synchronization and coordination problems.

Collection of nodes

- How to manage group membership?
- How to know that you are indeed communicating with an authorized (non)member?

Organization

Overlay network

Each node in the collection communicates only with other nodes in the system, its neighbors. The set of neighbors may be dynamic, or may even be known only implicitly (i.e., requires a lookup).

Overlay types

Well-known example of overlay networks: peer-to-peer systems.

Structured: each node has a well-defined set of neighbors with whom it can communicate (tree, ring).

Unstructured: each node has references to randomly selected other nodes from the system.

Coherent system

Essence

The collection of nodes as a whole operates the same, no matter where, when, and how interaction between a user and the system takes place.

Examples

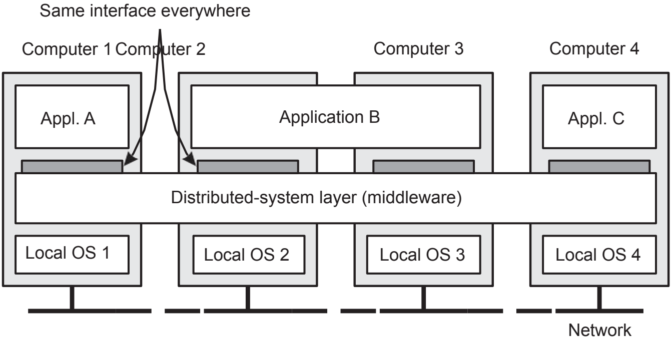
- An end user cannot tell where a computation is taking place
- Where data is exactly stored should be irrelevant to an application
- If or not data has been replicated is completely hidden

Keyword is distribution transparency

The snag: partial failures

It is inevitable that at any time only a part of the distributed system fails. Hiding partial failures and their recovery is often very difficult and in general impossible to hide.

Middleware: the OS of distributed systems



What does it contain?

Commonly used components and functions that need not be implemented by applications separately.

What do we want to achieve?

- 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

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

Observation

Aiming at full distribution transparency may be too much:

Degree of transparency

Observation

Aiming at full distribution transparency may be too much:

- There are communication latencies that cannot be hidden

Degree of transparency

Observation

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

Observation

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

Openness of distributed systems

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

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 and/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 and/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.

Size scalability

Root causes for scalability problems with centralized solutions

- The computational capacity, limited by the CPUs
- The storage capacity, including the transfer rate between CPUs and disks
- The network between the user and the centralized service

Problems with geographical scalability

- Cannot simply go from LAN to WAN: many distributed systems assume synchronous client-server interactions: client sends request and waits for an answer. Latency may easily prohibit this scheme.
- WAN links are often inherently unreliable: simply moving streaming video from LAN to WAN is bound to fail.
- Lack of multipoint communication, so that a simple search broadcast cannot be deployed. Solution is to develop separate naming and directory services (having their own scalability problems).

Problems with administrative scalability

Essence

Conflicting policies concerning usage (and thus payment), management, and security

Examples

- Computational grids: share expensive resources between different domains.
- Shared equipment: how to control, manage, and use a shared radio telescope constructed as large-scale shared sensor network?

Exception: several peer-to-peer networks

- File-sharing systems (based, e.g., on BitTorrent)
- Peer-to-peer telephony (Skype)
- Peer-assisted audio streaming (Spotify)

Note: end users collaborate and not administrative entities.

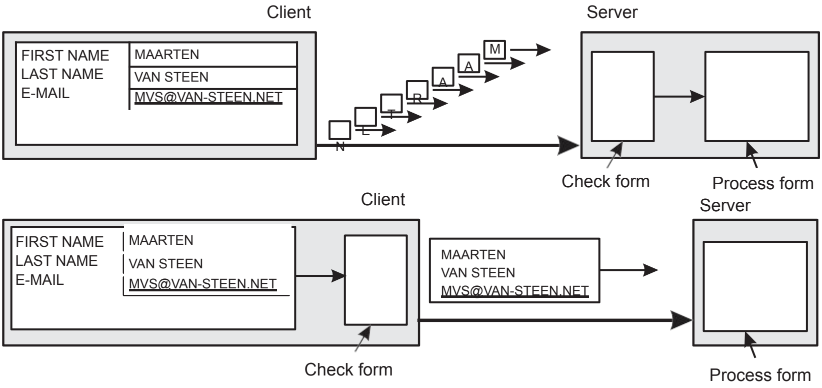
Techniques for scaling

Hide communication latencies

- Make use of asynchronous communication
- Have separate handler for incoming response
- Problem: not every application fits this model

Techniques for scaling

Facilitate solution by moving computations to client



Techniques for scaling

Partition data and computations across multiple machines

- Move computations to clients (Java applets)
- Decentralized naming services (DNS)
- Decentralized information systems (WWW)

Techniques for scaling

Replication and caching: Make copies of data available at different machines

- Replicated file servers and databases
- Mirrored Web sites
- Web caches (in browsers and proxies)
- File caching (at server and client)

Scaling: The problem with replication

Applying replication is easy, except for one thing

Scaling: The problem with replication

Applying replication is easy, except for one thing

- Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.

Scaling: The problem with replication

Applying replication is easy, except for one thing

- Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.

Scaling: The problem with replication

Applying replication is easy, except for one thing

- Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.
- Global synchronization precludes large-scale solutions.

Scaling: The problem with replication

Applying replication is easy, except for one thing

- Having multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
- Always keeping copies consistent and in a general way requires global synchronization on each modification.
- Global synchronization precludes large-scale solutions.

Observation

If we can tolerate inconsistencies, we may reduce the need for global synchronization, but tolerating inconsistencies is application dependent.

Developing distributed systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. Many false assumptions are often made.

Developing distributed systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. Many false assumptions are often made.

False (and often hidden) assumptions

Developing distributed systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. Many false assumptions are often made.

False (and often hidden) assumptions

- The network is reliable

Developing distributed systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. Many false assumptions are often made.

False (and often hidden) assumptions

- The network is reliable
- The network is secure

Developing distributed systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. Many false assumptions are often made.

False (and often hidden) assumptions

- The network is reliable
- The network is secure
- The network is homogeneous

Developing distributed systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. Many false assumptions are often made.

False (and often hidden) assumptions

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change

Developing distributed systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. Many false assumptions are often made.

False (and often hidden) assumptions

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero

Developing distributed systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. Many false assumptions are often made.

False (and often hidden) assumptions

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite

Developing distributed systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. Many false assumptions are often made.

False (and often hidden) 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

Developing distributed systems: Pitfalls

Observation

Many distributed systems are needlessly complex caused by mistakes that required patching later on. Many false assumptions are often made.

False (and often hidden) 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

Three types of distributed systems

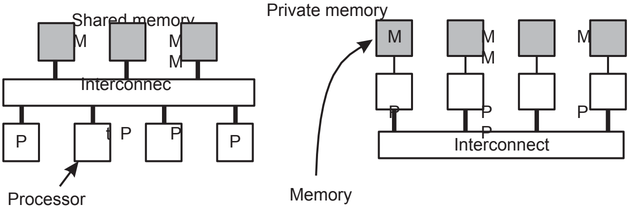
- High performance distributed computing systems
- Distributed information systems
- Distributed systems for pervasive computing

Parallel computing

Observation

High-performance distributed computing started with parallel computing

Multiprocessor and multicore versus multicomputer



Distributed shared memory systems

Observation

Multiprocessors are relatively easy to program in comparison to multicomputers, yet have problems when increasing the number of processors (or cores). Solution: Try to implement a shared-memory model on top of a multicomputer.

Example through virtual-memory techniques

Map all main-memory pages (from different processors) into one single virtual address space. If process at processor *A* addresses a page *P* located at processor *B*, the OS at *A* traps and fetches *P* from *B*, just as it would if *P* had been located on local disk.

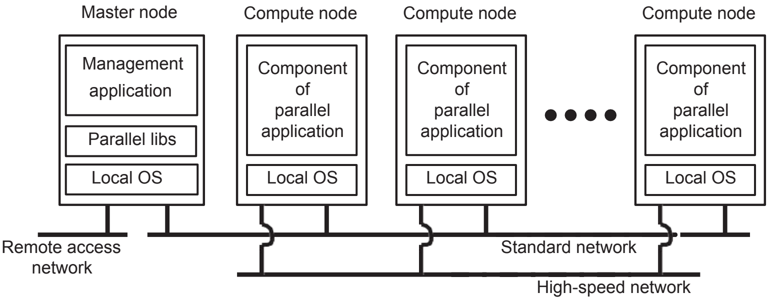
Problem

Performance of distributed shared memory could never compete with that of multiprocessors, and failed to meet the expectations of programmers. It has been widely abandoned by now.

Cluster computing

Essentially a group of high-end systems connected through a LAN

- Homogeneous: same OS, near-identical hardware
- Single managing node



Grid computing

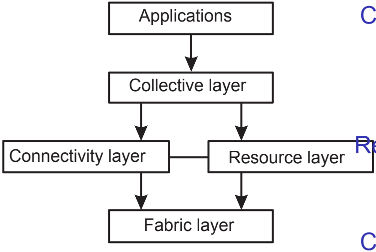
The next step: lots of nodes from everywhere

- Heterogeneous
- Dispersed across several organizations
- Can easily span a wide-area network

Note

To allow for collaborations, grids generally use virtual organizations. In essence, this is a grouping of users (or better: their IDs) that will allow for authorization on resource allocation.

Architecture for grid computing



The layers

Fabric: Provides interfaces to local resources (for querying state and capabilities, locking, etc.)

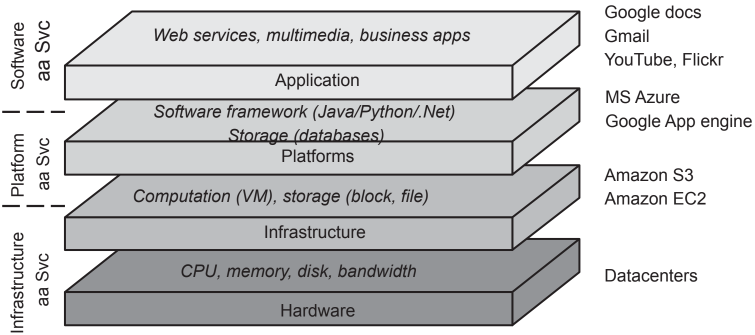
Connectivity: Communication/transaction protocols, e.g., for moving data between resources. Also various authentication protocols.

Resource: Manages a single resource, such as creating processes or reading data.

Collective: Handles access to multiple resources: discovery, scheduling, replication.

Application: Contains actual grid applications in a single organization.

Cloud computing



Cloud computing

Make a distinction between four layers

- Hardware: Processors, routers, power and cooling systems. Customers normally never get to see these.
- Infrastructure: Deploys virtualization techniques. Evolves around allocating and managing virtual storage devices and virtual servers.
- Platform: Provides higher-level abstractions for storage and such. Example: Amazon S3 storage system offers an API for (locally created) files to be organized and stored in so-called buckets.
- Application: Actual applications, such as office suites (text processors, spreadsheet applications, presentation applications). Comparable to the suite of apps shipped with OSes.