Distributed Systems
Principles and Paradigms

Maarten van Steen

VU Amsterdam, Dept. Computer Science
Room R4.20, steen@cs.vu.nl

Chapter 01: Introduction

Version: February 21, 2011

vrije Universiteit amsterdam

Contents

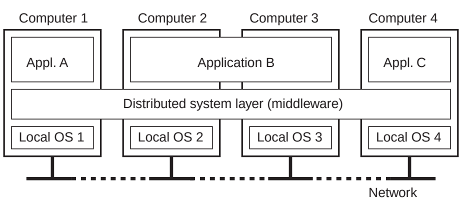
2 / 26

Distributed System: Definition

A distributed system is a piece of software that ensures that:

> *a collection of independent computers appears to its users as a single coherent system*

Two aspects: (1) independent computers and (2) single system ⇒ middleware.



3 / 26

Goals of Distributed Systems

- Making resources available
- Distribution transparency
- Openness
- Scalability

Distribution Transparency

Transp.	Description
Access	Hides differences in data representation and invocation mechanisms
Location	Hides where an object resides
Migration	Hides from an object the ability of a system to change that object's location
Relocation	Hides from a client the ability of a system to change the location of an object to which the client is bound
Replication	Hides the fact that an object or its state may be replicated and that replicas reside at different locations
Concurrency	Hides the coordination of activities between objects to achieve consistency at a higher level
Failure	Hides failure and possible recovery of objects

Note

Distribution transparency is a nice a goal, but achieving it is a different story.

Degree of Transparency

Observation

Aiming at full distribution transparency may be too much:

- Users may be located in different continents
- 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 Web caches exactly up-to-date with the master
  - Immediately flushing write operations to disk for fault tolerance

Openness of Distributed Systems

Open distributed system

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

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

Achieving openness

At least make the distributed system independent from heterogeneity of the underlying environment:

- Hardware
- Platforms
- Languages

Policies versus Mechanisms

Implementing openness

Requires support for different 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

Ideally, a distributed system provides only 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

Scale in Distributed Systems

Observation

Many developers of modern distributed system easily use the adjective “scalable” without making clear **why** their system actually scales.

Scalability

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. The (non)solution: powerful servers. Today, the challenge lies in geographical and administrative scalability.

Techniques for Scaling

Hide communication latencies

Avoid waiting for responses; do something else:

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

Introduction 1.2 Goals

10 / 26

Techniques for Scaling

Distribution

Partition data and computations across multiple machines:

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

Introduction 1.2 Goals

11 / 26

Techniques for Scaling

Replication/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)

Introduction 1.2 Goals

12 / 26

Scaling – The Problem

Observation

Applying scaling techniques 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. There are many 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

- Distributed Computing Systems
- Distributed Information Systems
- Distributed Pervasive Systems

Distributed Computing Systems

Observation

Many distributed systems are configured for High-Performance Computing

Cluster Computing

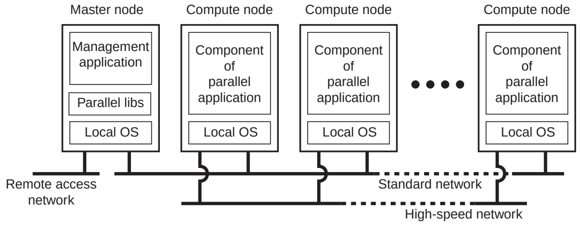
Essentially a group of high-end systems connected through a LAN:

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

Introduction 1.3 Types of Distributed Systems

16 / 26

Distributed Computing Systems



Introduction 1.3 Types of Distributed Systems

17 / 26

Distributed Computing Systems

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.

Introduction 1.3 Types of Distributed Systems

18 / 26

Distributed Information Systems

Observation

The vast amount of distributed systems in use today are forms of traditional information systems, that now integrate legacy systems.
Example: Transaction processing systems.

```
BEGIN_TRANSACTION(server, transaction)
READ(transaction, file-1, data)
WRITE(transaction, file-2, data)
newData := MODIFIED(data)
IF WRONG(newData) THEN
    ABORT_TRANSACTION(transaction)
ELSE
    WRITE(transaction, file-2, newData)
    END_TRANSACTION(transaction)
END IF
```

Note

Transactions form an atomic operation.

Introduction 1.3 Types of Distributed Systems

Distributed Information Systems: Transactions

Model

A transaction is a collection of operations on the state of an object (database, object composition, etc.) that satisfies the following properties (ACID)

Atomicity: All operations either succeed, or all of them fail. When the transaction fails, the state of the object will remain unaffected by the transaction.

Consistency: A transaction establishes a valid state transition. This does not exclude the possibility of invalid, intermediate states during the transaction's execution.

Isolation: Concurrent transactions do not interfere with each other. It appears to each transaction *T* that other transactions occur either before *T*, or after *T*, but never both.

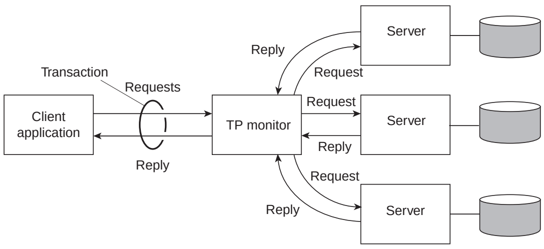
Durability: After the execution of a transaction, its effects are made permanent: changes to the state survive failures.

Introduction 1.3 Types of Distributed Systems

Transaction Processing Monitor

Observation

In many cases, the data involved in a transaction is distributed across several servers. A TP Monitor is responsible for coordinating the execution of a transaction

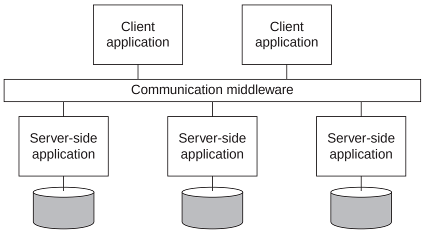


Introduction 1.3 Types of Distributed Systems

Distr. Info. Systems: Enterprise Application Integration

Problem

A TP monitor doesn't separate apps from their databases. Also needed are facilities for direct communication between apps.



- Remote Procedure Call (RPC)
- Message-Oriented Middleware (MOM)

Distributed Pervasive Systems

Observation

Emerging next-generation of distributed systems in which nodes are small, mobile, and often embedded in a larger system.

Some requirements

- Contextual change: The system is part of an environment in which changes should be immediately accounted for.
- Ad hoc composition: Each node may be used in a very different ways by different users. Requires ease-of-configuration.
- Sharing is the default: Nodes come and go, providing sharable services and information. Calls again for simplicity.

Note

Pervasiveness and distribution transparency: a good match?

Pervasive Systems: Examples

Home Systems

Should be completely self-organizing:

- There should be no system administrator
- Provide a personal space for each of its users
- Simplest solution: a centralized home box?

Electronic health systems

Devices are physically close to a person:

- Where and how should monitored data be stored?
- How can we prevent loss of crucial data?
- What is needed to generate and propagate alerts?
- How can security be enforced?
- How can physicians provide online feedback?

Sensor networks

Characteristics

The nodes to which sensors are attached are:

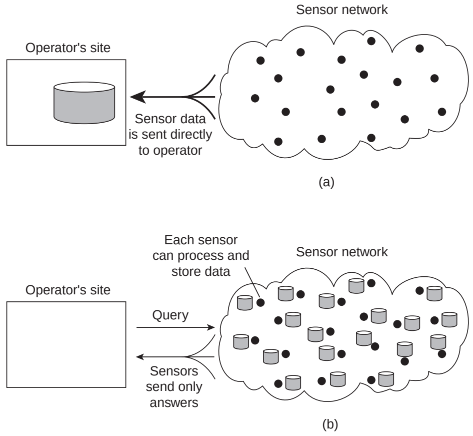
- Many (10s-1000s)
- Simple (small memory/compute/communication capacity)
- Often battery-powered (or even battery-less)

Introduction 1.3 Types of Distributed Systems

25 / 26

Introduction 1.3 Types of Distributed Systems

Sensor networks as distributed systems



Introduction 1.3 Types of Distributed Systems

26 / 26