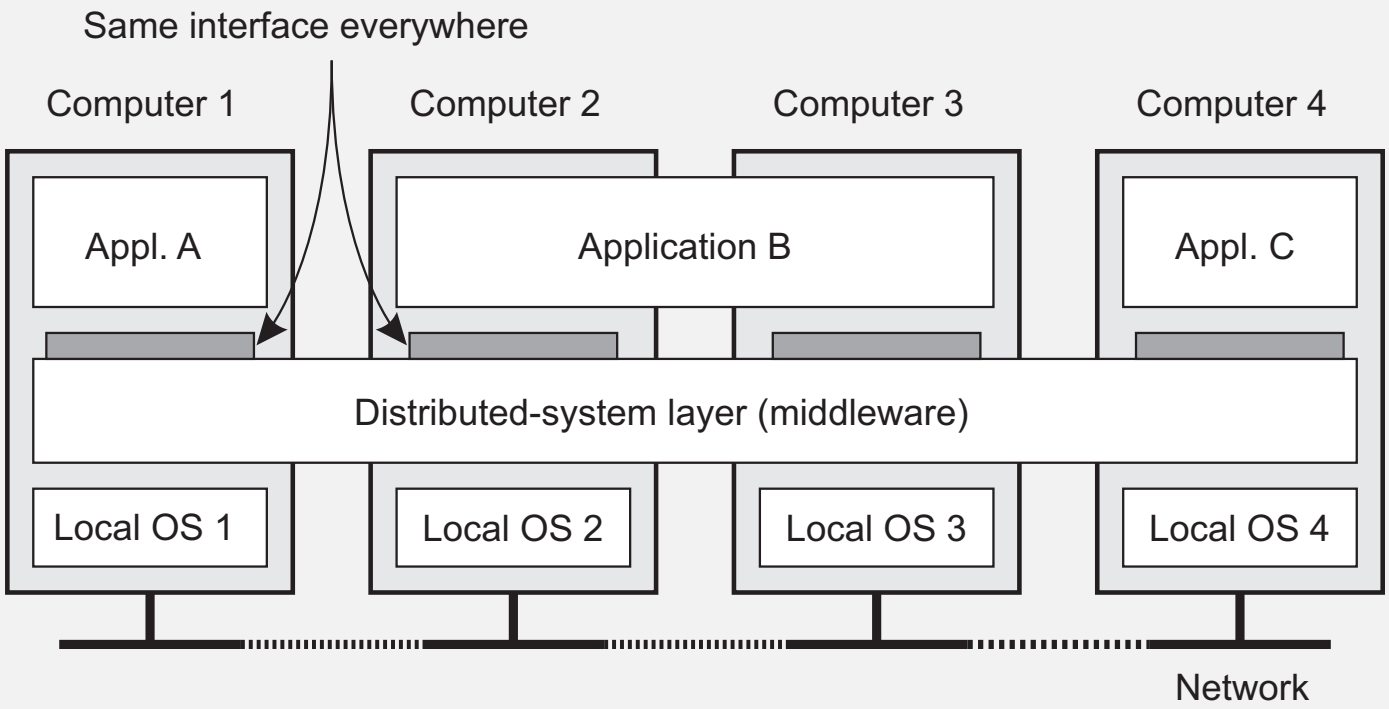
Distributed System: Definition

A distributed system is

> *a collection of autonomous computing elements that appears to its users as a single coherent system*

Two aspects: (1) independent computing elements and (2) single system $\Rightarrow$ middleware.



Goals of Distributed Systems

- Making resources available
- Distribution transparency
- Openness
- Scalability

Distribution transparency

Transp.	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

Note

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

Distribution transparency

Transp.	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

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

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

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

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

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

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.

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.

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

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)

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)

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.

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.

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.

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.

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

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

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

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

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

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

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

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

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

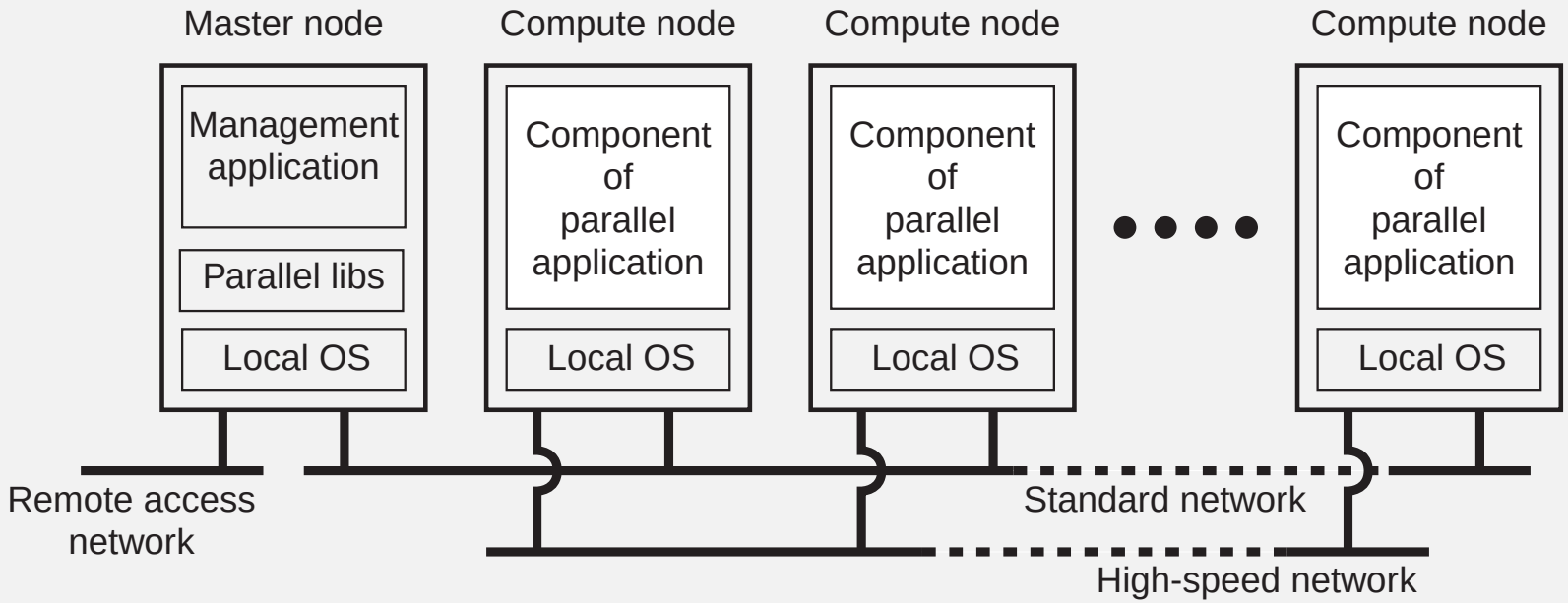
Many distributed systems are configured for High-Performance Computing

Cluster Computing

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

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

Distributed computing systems



Distributed computing systems

Grid Computing

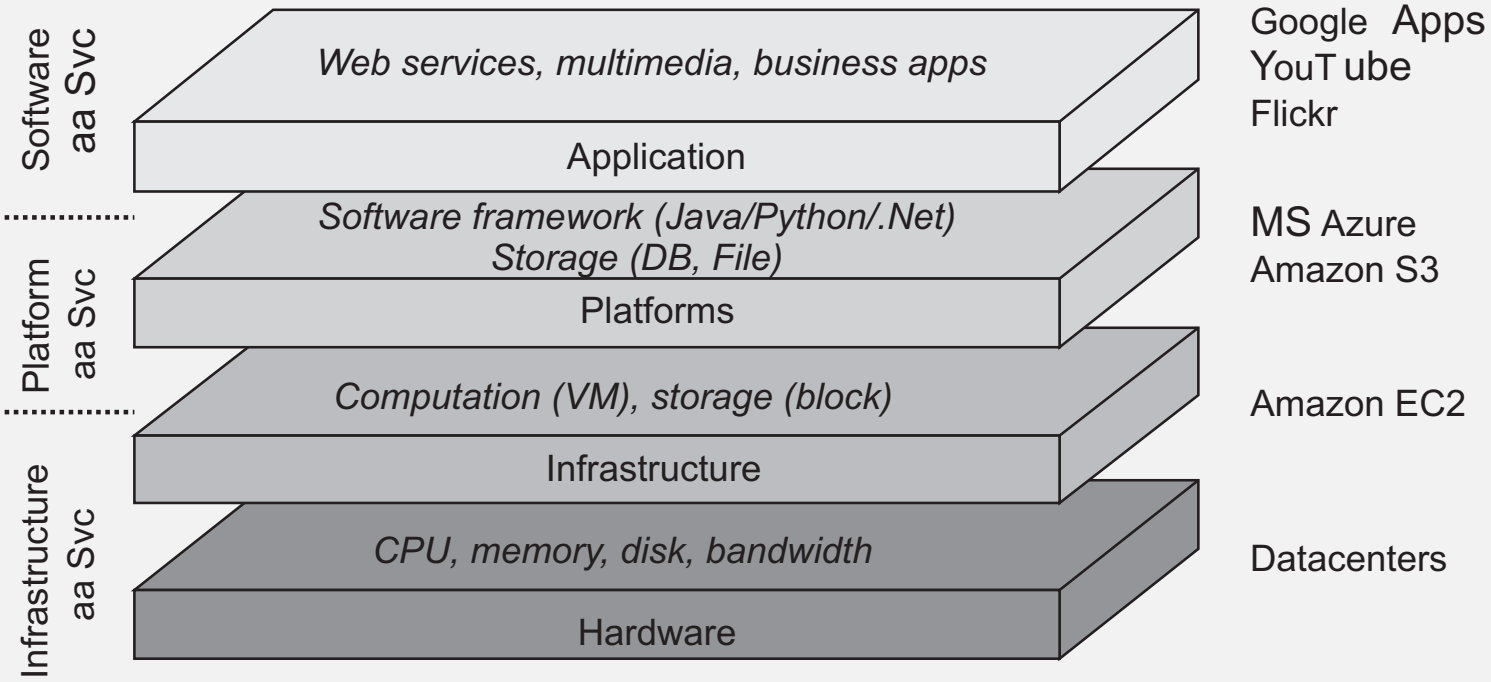
The next step: lots of nodes from everywhere:

- 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.

Distributed computing systems: Clouds



Distributed computing systems: Clouds

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.

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
```

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.

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.

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.

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.

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.

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.

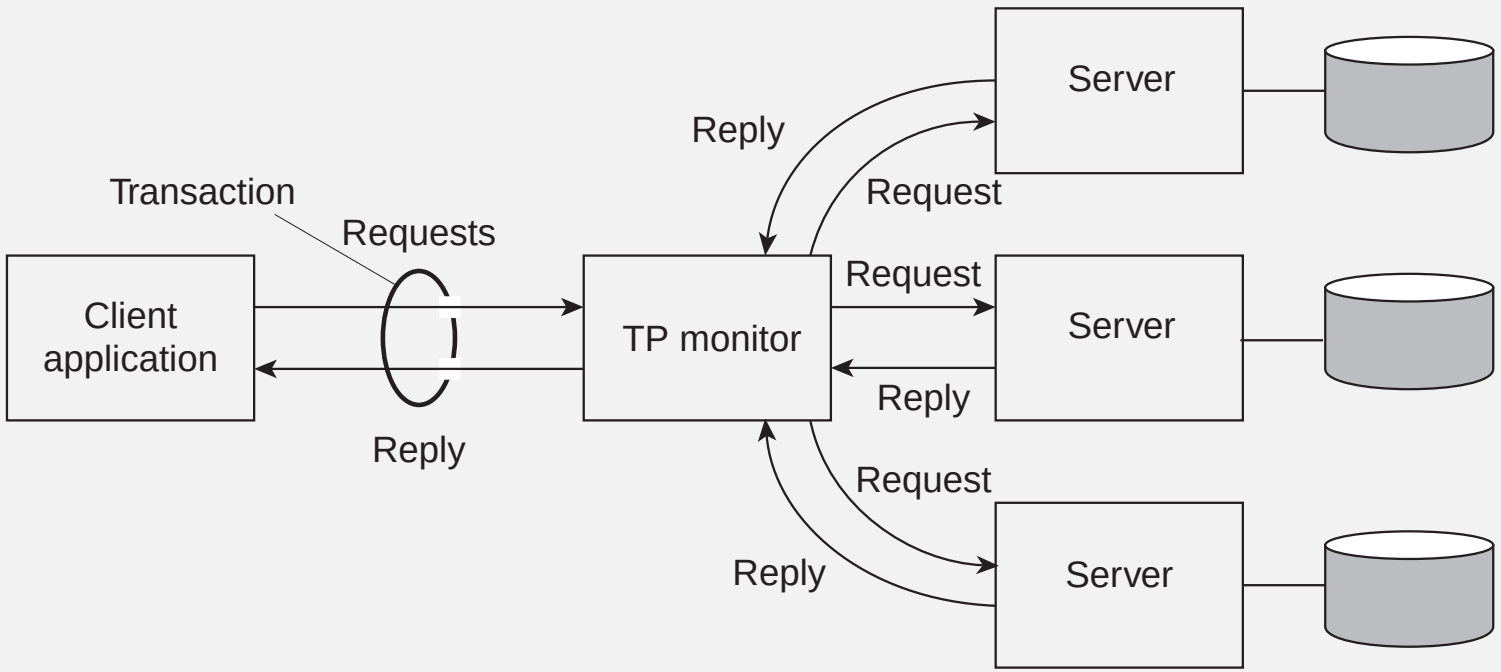
Isolation: Concurrent transactions do not interfere with each other. It appears to each transaction T that other transactions occur either before T, or after T, but never both.

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

Transaction processing monitor

Observation

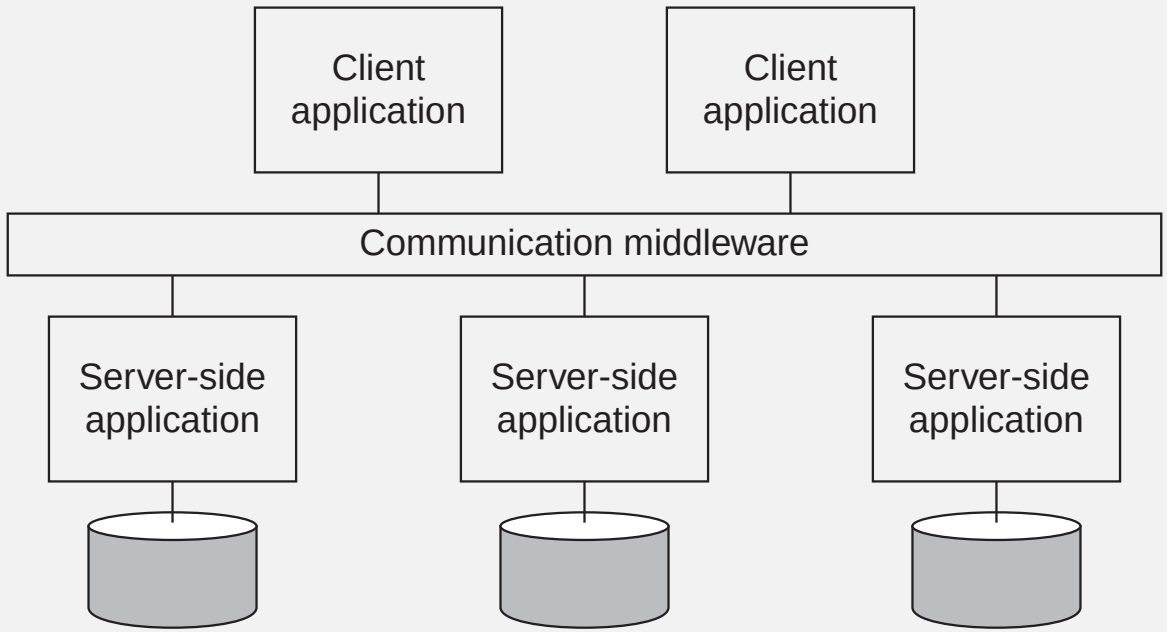
In many cases, the data involved in a transaction is distributed across several servers. A TP Monitor is responsible for coordinating the execution of a transaction



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, characterized by the fact that the system naturally blends into the user's environment.

Three (overlapping) subtypes

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

Distributed pervasive systems

Observation

Emerging next-generation of distributed systems in which nodes are small, mobile, and often embedded in a larger system, characterized by the fact that the system naturally blends into the user's environment.

Three (overlapping) subtypes

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

Distributed pervasive systems

Observation

Emerging next-generation of distributed systems in which nodes are small, mobile, and often embedded in a larger system, characterized by the fact that the system naturally blends into the user's environment.

Three (overlapping) subtypes

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

Distributed pervasive systems

Observation

Emerging next-generation of distributed systems in which nodes are small, mobile, and often embedded in a larger system, characterized by the fact that the system naturally blends into the user's environment.

Three (overlapping) subtypes

- Ubiquitous computing systems: pervasive and continuously present, i.e., there is a continous interaction between system and user.
- Mobile computing systems: pervasive, but emphasis is on the fact that devices are inherently mobile.
- Sensor (and actuator) 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 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

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 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

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 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

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 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

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 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

Observation

Mobile computing systems are generally a subclass of ubiquitous computing systems and meet all of the five requirements.

Typical characteristics

- Many different types of mobile divices: smart phones, remote controls, car equipment, and so on
- Wireless communication
- Devices may continuously change their location $\Rightarrow$
  - setting up a route may be problematic, as routes can change frequently
  - devices may easily be temporarily disconnected $\Rightarrow$ disruption-tolerant networks

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)