Introduction to Distributed Systems

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Based on slides by Maarten Van Steen

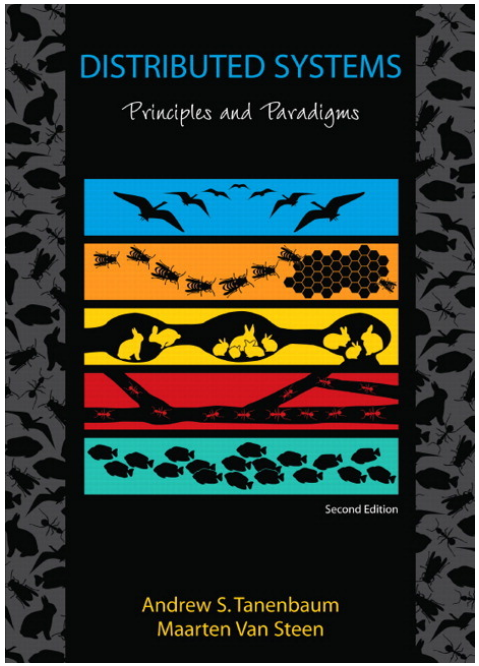
Course Information

Course Objective

- The course covers fundamental models for distributed systems:
  - Inter process communication
  - Synchronization
  - Consistency
  - Replication
  - Fault tolerance
  - Security

Course Material

- Distributed Systems: Principles and Paradigms, 2nd edition, 2007
  Andrew S. Tanenbaum and Maarten van Steen



Course Examination

- Mid term exam: 20%
- Final exam: 20%
- Review questions: 10 assignments 20%
- Lab assignments: four assignments in Java 40%
- The assignments should be done in group of two.

Distributed Systems
Definition and Goals

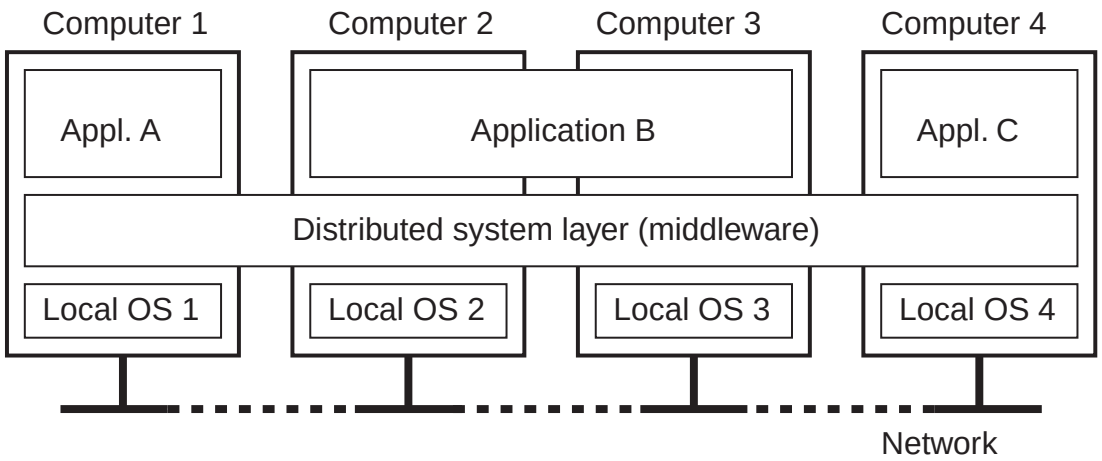
Distributed System

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.

Distributed System

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
  2. Single system $\Rightarrow$ middleware



Goals of Distributed Systems

- Making resources available
- Distribution transparency
- Openness
- Scalability

Resource Availability

Making Resources Available

- Make it easy for users to

Making Resources Available

- Make it easy for users to
  - Access remote resources

Making Resources Available

- Make it easy for users to
  - Access remote resources
  - Share resources

Making Resources Available

- Make it easy for users to
  - Access remote resources
  - Share resources
- Resources: printer, computers, storage, data, files, ...

Distribution Transparency

Distribution Transparency

- Transparency:

Distribution Transparency

- Transparency:
  - Access $\rightarrow$ hide differences in data representation and invocation mechanisms.

Distribution Transparency

- Transparency:
  - Access → hide differences in data representation and invocation mechanisms.
  - Location → hide where an object is located.

Distribution Transparency

- Transparency:
  - Access → hide differences in data representation and invocation mechanisms.
  - Location → hide where an object is located.
  - Relocation → hide that an object may be moved to another location while in use.

Distribution Transparency

- Transparency:
  - Access → hide differences in data representation and invocation mechanisms.
  - 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.

Distribution Transparency

- Transparency:
  - Access → hide differences in data representation and invocation mechanisms.
  - 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.

Distribution Transparency

- Transparency:
  - Access → hide differences in data representation and invocation mechanisms.
  - 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.

Distribution Transparency

- Transparency:
  - Access → hide differences in data representation and invocation mechanisms.
  - 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 failure and possible recovery of an object.

Distribution Transparency

- Transparency:
  - Access → hide differences in data representation and invocation mechanisms.
  - 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 failure and possible recovery of an object.
- Distribution transparency is a nice goal, but achieving it is a different story.

Degree of Transparency

- Distribution transparency is generally considered preferable, but...

Degree of Transparency

- Distribution transparency is generally considered preferable, but...
- In some situations, hiding all distribution aspects from users is not a good idea.

Degree of Transparency

- Distribution transparency is generally considered preferable, but...
- In some situations, hiding all distribution aspects from users is not a good idea.
- Trade-off between a high degree of transparency and the performance of a system.

Degree of Transparency

- Distribution transparency is generally considered preferable, but...
- In some situations, hiding all distribution aspects from users is not a good idea.
- Trade-off between a high degree of transparency and the performance of a system.
- Completely hiding failures of networks and nodes is impossible.
  - You cannot distinguish a slow computer from a failed one.

Openness

Openness of Distributed Systems

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

Openness of Distributed Systems

- Be able to interact with services from other open systems, irrespective of the underlying environment.
  - Conforms to well-defined interfaces

Openness of Distributed Systems

- Be able to interact with services from other open systems, irrespective of the underlying environment.
  - Conforms to well-defined interfaces
  - Supports portability of applications

Openness of Distributed Systems

- Be able to interact with services from other open systems, irrespective of the underlying environment.
  - Conforms to well-defined interfaces
  - Supports portability of applications
  - Interoperate easily

Openness of Distributed Systems

- Be able to interact with services from other open systems, irrespective of the underlying environment.
  - Conforms to well-defined interfaces
  - Supports portability of applications
  - Interoperate easily
- Independent from heterogeneity of the underlying environment:
  - E.g., hardware, platforms, languages, ...

Implementing Openness - Policies vs. Mechanisms

- Requires support for different policies.

Implementing Openness - Policies vs. Mechanisms

- Requires support for different policies.
  - For example:
  - What level of consistency do we require for client-cached data?

Implementing Openness - Policies vs. Mechanisms

- Requires support for different policies.
  - For example:
  - What level of consistency do we require for client-cached data?
  - Which operations do we allow downloaded code to perform?

Implementing Openness - Policies vs. Mechanisms

- Requires support for different policies.
  - For example:
  - 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?

Implementing Openness - Policies vs. Mechanisms

- Requires support for different policies.
  - For example:
  - 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?

- Ideally, a distributed system provides only mechanisms.

Implementing Openness - Policies vs. Mechanisms

- Requires support for different policies.
  - For example:
  - 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?

- Ideally, a distributed system provides only mechanisms.
  - For example:
  - Support different levels of trust for mobile code.

Implementing Openness - Policies vs. Mechanisms

- Requires support for different policies.
  - For example:
  - 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?

- Ideally, a distributed system provides only mechanisms.
  - For example:
  - Support different levels of trust for mobile code.
  - Provide adjustable QoS parameters per data stream.

Implementing Openness - Policies vs. Mechanisms

- Requires support for different policies.
  - For example:
  - 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?

- Ideally, a distributed system provides only mechanisms.
  - For example:
  - Support different levels of trust for mobile code.
  - Provide adjustable QoS parameters per data stream.
  - Offer different encryption algorithms.

Scalability

Scale in Distributed Systems

- Scalability of at least three components:

Scale in Distributed Systems

- Scalability of at least three components:
  - (Size scalability): number of users and/or processes

Scale in Distributed Systems

- Scalability of at least three components:
  - (Size scalability): number of users and/or processes
  - (Geographical scalability): maximum distance between nodes

Scale in Distributed Systems

- Scalability of at least three components:
  - (Size scalability): number of users and/or processes
  - (Geographical scalability): maximum distance between nodes
  - (Administrative scalability): number of administrative domains

Techniques for Scaling

- Hide communication latencies: avoid waiting for responses; do something else.
  - E.g., make use of asynchronous communication

Techniques for Scaling

- Hide communication latencies: avoid waiting for responses; do something else.
  - E.g., make use of asynchronous communication

- Distribution: partition data and computations across multiple machines.
  - E.g., move computations to clients (Java applets), decentralized naming services (DNS), decentralized information systems (WWW)

Techniques for Scaling

- Hide communication latencies: avoid waiting for responses; do something else.
  - E.g., make use of asynchronous communication
- Distribution: partition data and computations across multiple machines.
  - E.g., move computations to clients (Java applets), decentralized naming services (DNS), decentralized information systems (WWW)
- Replication/caching: make copies of data available at different machines.
  - E.g., replicated file servers and databases

Scaling – The Problem

- Applying scaling techniques is easy, except for one thing:

Scaling – The Problem

- Applying scaling techniques is easy, except for one thing:
  - Multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.

Scaling – The Problem

- Applying scaling techniques is easy, except for one thing:
  - Multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
  - Global synchronization on each modification.

Scaling – The Problem

- Applying scaling techniques is easy, except for one thing:
  - Multiple copies (cached or replicated), leads to inconsistencies: modifying one copy makes that copy different from the rest.
  - Global synchronization on each modification.

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

Developing Distributed Systems: Pitfalls

- 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

Types of Distributed Systems

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

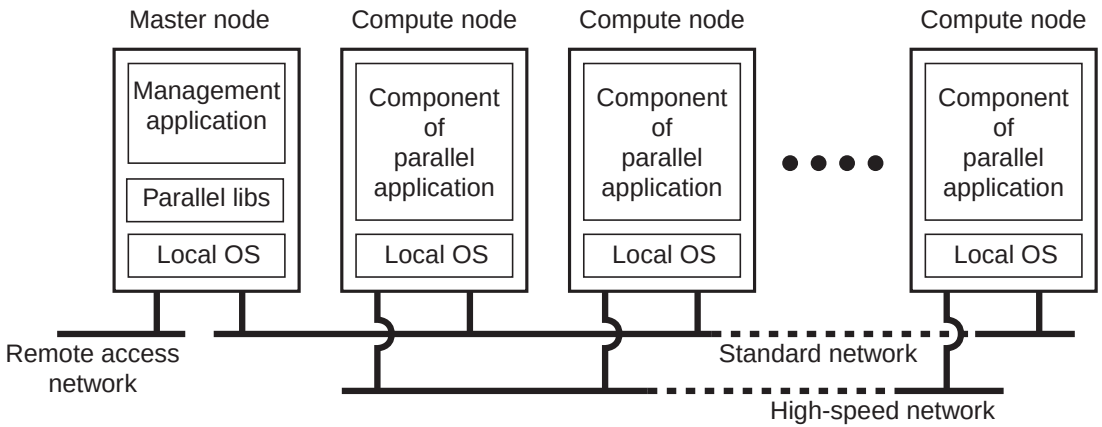
Distributed Computing Systems

Distributed Computing Systems: Cluster

- Many distributed systems are configured for High-Performance Computing.

Distributed Computing Systems: Cluster

- Many distributed systems are configured for High-Performance Computing.
- Cluster computing: a group of high-end systems connected through a LAN.
  - Homogeneous: same OS, near-identical hardware
  - Single managing node



Distributed Computing Systems: Grid

- Grid computing: lots of nodes from everywhere.
  - Heterogeneous
  - Dispersed across several organizations
  - Can easily span a wide-area network

Distributed Computing Systems: Grid

- Grid computing: lots of nodes from everywhere.
  - Heterogeneous
  - Dispersed across several organizations
  - Can easily span a wide-area network

- To allow for collaborations, grids generally use virtual organizations.
  - A group of users that will allow for authorization on resource allocation.

Distributed Computing Systems: Cloud (1/2)

- Cloud computing: make a distinction between four layers.

Distributed Computing Systems: Cloud (1/2)

- Cloud computing: make a distinction between four layers.
  - Hardware: processors, routers, power and cooling systems. Customers normally never get to see these.

Distributed Computing Systems: Cloud (1/2)

- 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. (IaaS)

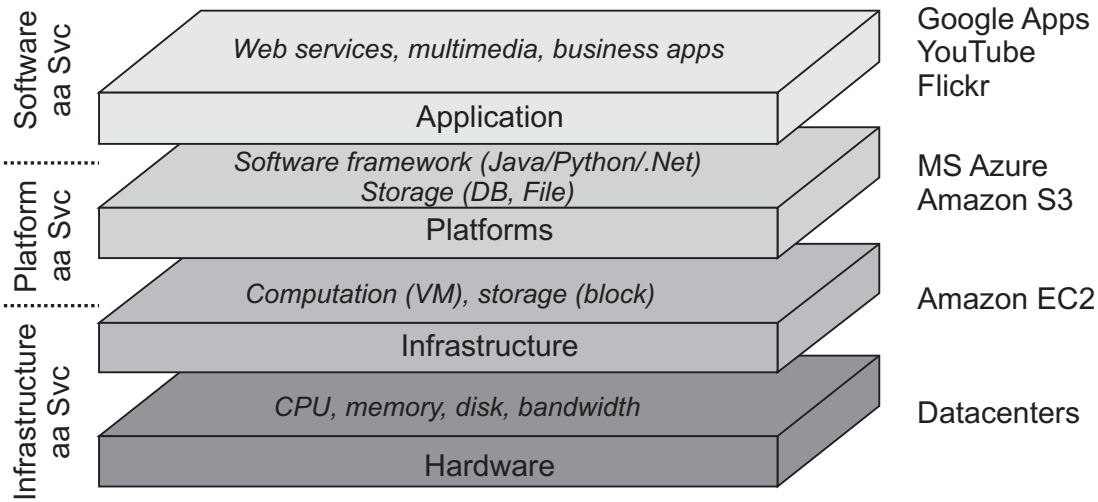
Distributed Computing Systems: Cloud (1/2)

- 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. (IaaS)
  - Platform: provides higher-level abstractions for storage and such. (PaaS)

Distributed Computing Systems: Cloud (1/2)

- 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. (IaaS)
  - Platform: provides higher-level abstractions for storage and such. (PaaS)
  - Application: actual applications, such as office suites, e.g., text processors, spreadsheet applications. (SaaS)

Distributed Computing Systems: Cloud (2/2)



Distributed Information Systems

Distributed Information Systems

- 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
```

- Transactions form an atomic operation.

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

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

- Atomicity: all operations either succeed, or all of them fail.

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

- Atomicity: all operations either succeed, or all of them fail.
- Consistency: a transaction establishes a valid state transition.

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

- Atomicity: all operations either succeed, or all of them fail.
- Consistency: a transaction establishes a valid state transition.
- Isolation: concurrent transactions do not interfere with each other.

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

- Atomicity: all operations either succeed, or all of them fail.
- Consistency: a transaction establishes a valid state transition.
- Isolation: concurrent transactions do not interfere with each other.
- Durability: after the execution of a transaction, its effects are made permanent.

Distributed Pervasive Systems

Distributed Pervasive Systems

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

Distributed Pervasive Systems

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

Distributed Pervasive Systems

- 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.

Distributed Pervasive Systems

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

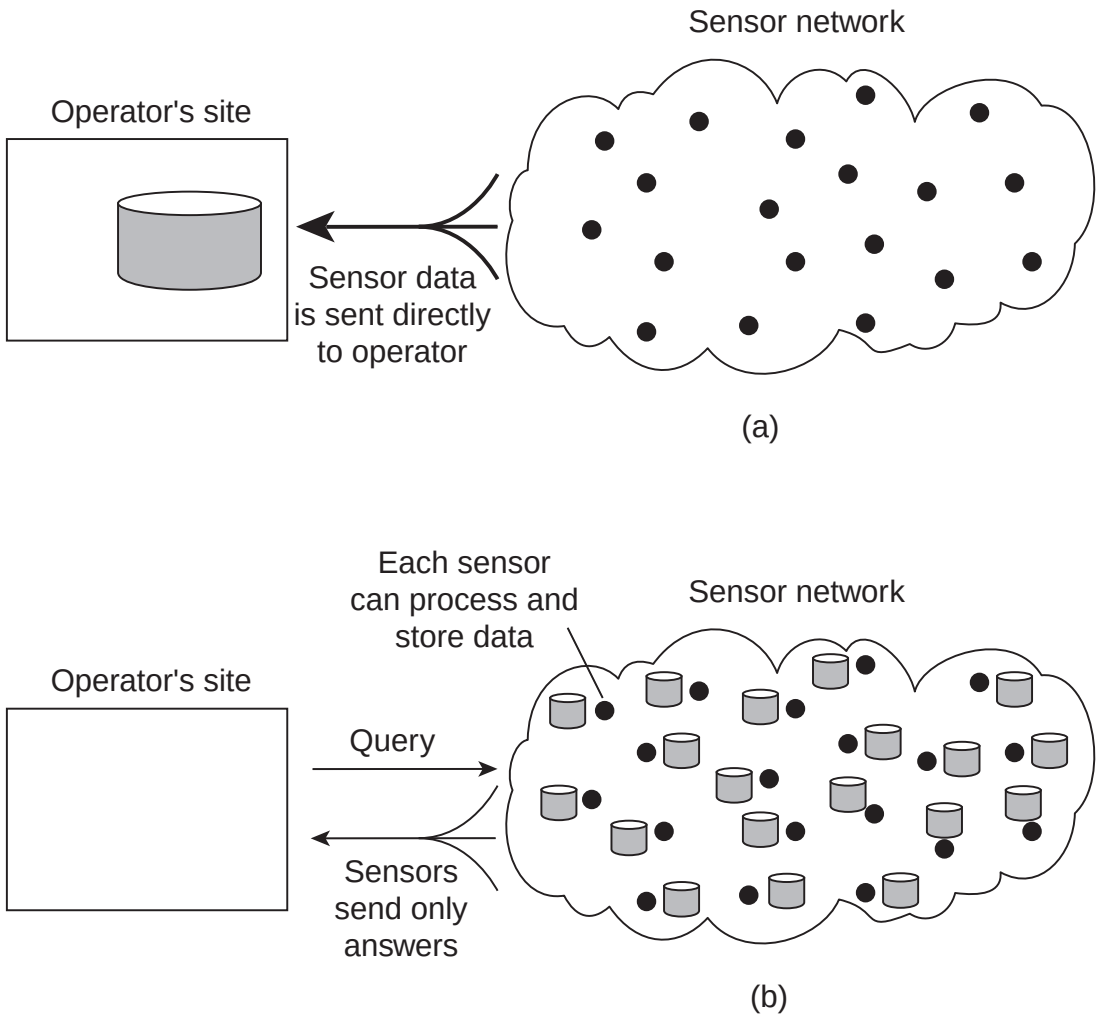
- Some requirements:
  - Contextual change: the system is part of an environment in which changes should be immediately accounted for.
  - Ad hoc composition: each node may be used in a very different ways by different users.

Distributed Pervasive Systems

- 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.
  - Sharing is the default: nodes come and go, providing sharable services and information.

Sensor networks



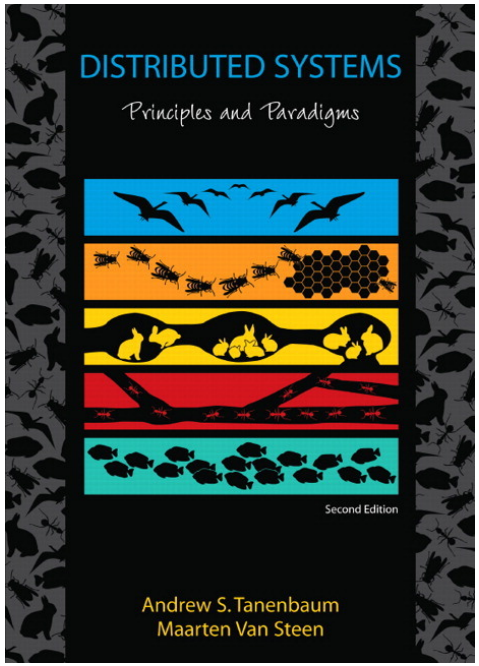
Summary

Summary

- Distributed system goals
  - Making resources available
  - Distribution transparency
  - Openness
  - Scalability
- Types of distributed systems
  - Distributed computing systems: cluster, grid, cloud
  - Distributed information systems: transaction (ACID)
  - Distributed pervasive systems: sensor networks

Reading

- Chapter 1 of the Distributed Systems: Principles and Paradigms.



Questions?