Lecture#1: Introduction to Distributed Systems

S. M. Ahsan Kazmi

Outline

- Introduction
  - What, why, advantages & challenges?
  - Basics of distributed systems

What are Distributed System?

- A distributed system:
  - Multiple connected CPUs working together
  - A collection of independent computers that appears to its users as a single coherent system
- Examples: parallel machines, networked machines

Why Distributed Systems?

- Many systems that we use on a daily basis are distributed
  - World wide web, Google
  - Amazon.com
  - Peer-to-peer file sharing systems
  - Grid and cluster computing
  - Modern networked computers

- Useful to understand how such real-world systems work

- Course covers basic principles for designing distributed systems

Advantages and Disadvantages

- Advantages
  - Communication and resource sharing possible
  - Economics – price-performance ratio
  - Reliability
  - Scalability
  - Potential for incremental growth

- Disadvantages
  - Distribution-aware Programming Languages, OSs and applications
  - Network connectivity essential
  - Security and privacy

Goals of Distributed Systems

- Making resource accessible or sharable
- Distribution transparency
- Openness
- Scalability

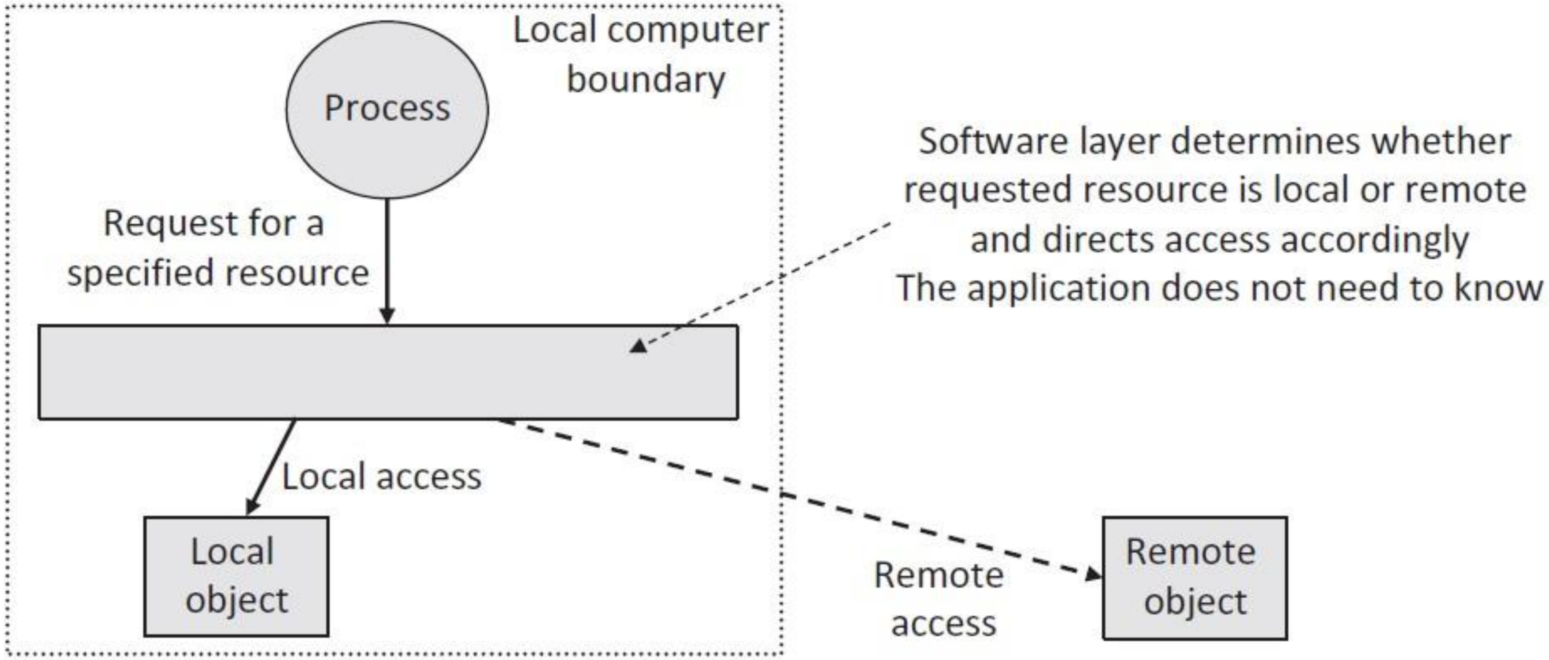
Transparency in a Distributed System

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be shared by several competitive users
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Different forms of transparency in a distributed system.

Access Transparency

- Access transparency requires that objects (this includes resources and services) are accessed with the same operations regardless of whether they are local or remote.



Location Transparency

- Location transparency is the ability to access objects without the knowledge of their location

- The provision of location transparency is often achieved through the use of special services whose role is to perform a mapping between a resource's name and its address

- Resource virtualization ("access transparency") – also provides location transparency

Relocation Transparency

- Hide that an object may be moved to another location while in use

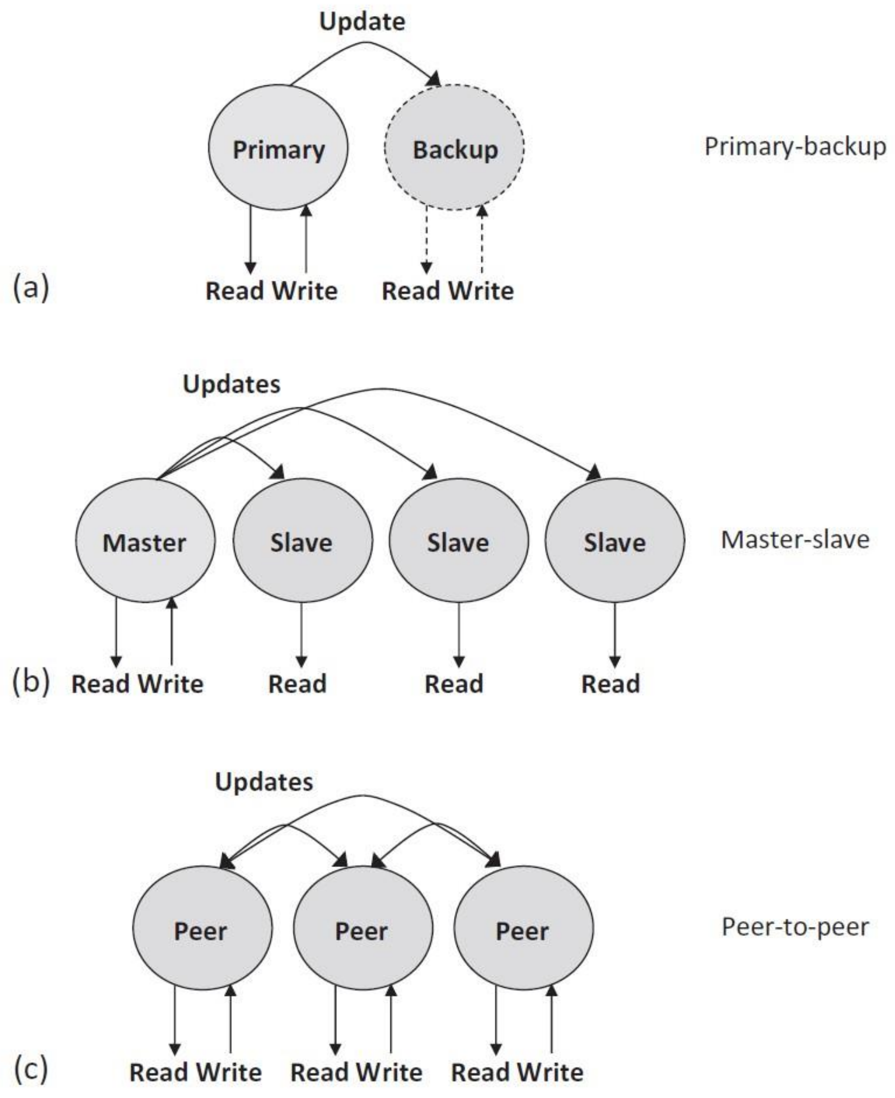
- The system may decide to move data to another location for performance optimization reasons

- Example: System automatically moves an object to another location due to high latency in current location

Replication Transparency

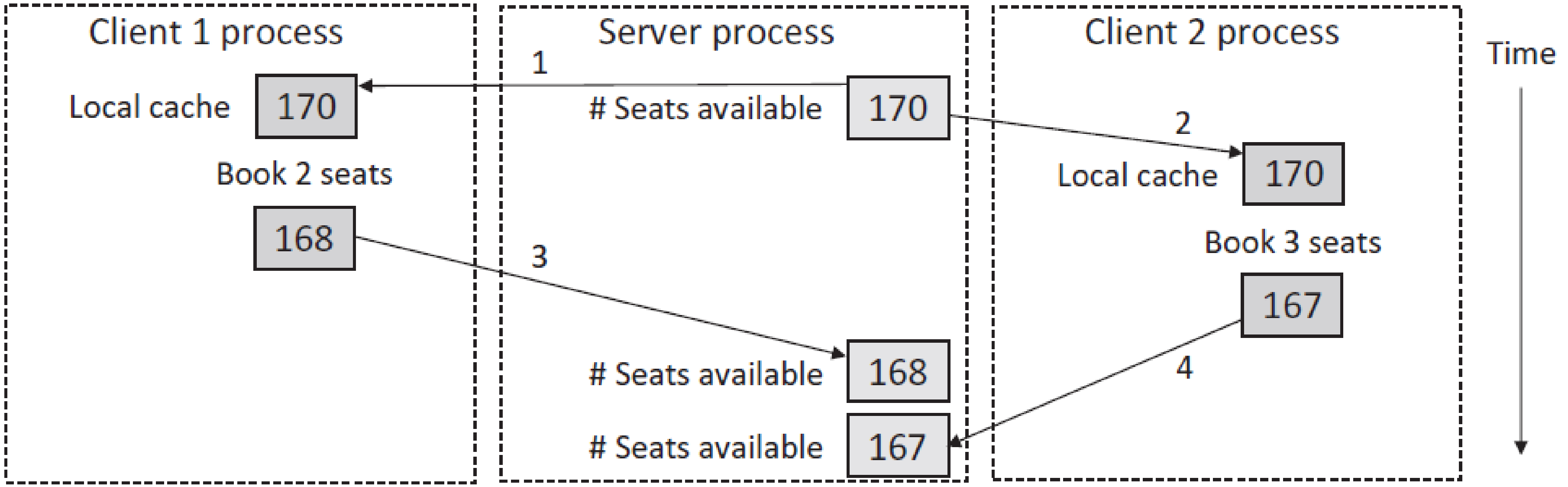
- Requires that multiple copies of objects can be created without any effect of the replication seen by applications that uses these objects

- The most significant challenge – maintenance of consistency



Concurrency Transparency

- Requires that concurrent processes can share objects without interference
- Raises the issue of data consistency
- Example of lost update problem – an airline seat booking scenario:



Failure Transparency

- Failure transparency requires that faults are concealed so that applications can continue to function without any impact on behavior or correctness

- Failures in distributed systems are inevitable

- We need to rely on runtime techniques that will deal with the failures

- Example – TCP and UDP
  - TCP has a number of built-in features that transparently deal with various failure
  - UDP is a more lightweight protocol – “send and pray”

Degree of Transparency

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

Open Distributed Systems

- Offer services that are described a priori
  - Syntax and semantics are known via protocols
- Services specified via interfaces
- Benefits
  - Interoperability
  - Portability
  - Extensibility
    - Open system evolve over time and should be extensible to accommodate new functionality.

Scalability

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

Dimensions of scalability:

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

Most systems account only, to a certain extent, for size scalability.

Scalability (cont.)

Limitation of size scalability

- Centralized services
- Centralized data
- Centralized algorithms

Scalability (cont.)

Limitations of geographical scalability

- Synchronous communication
- Networks is unreliable
- Centralized components in the system

Scalability (cont.)

Limitations of administrative scalability

- Conflicting policies
  - Resource usage
  - Management
  - Security

Scaling Techniques

Hide communication latencies – avoid waiting for responses; do something else:

- Make use of asynchronous communication
- Problem: not every application fits this model

Scaling Techniques (cont.)

Distribution – partition data and computations across multiple machines:

- Move computations to clients
- Decentralized naming services
- Decentralized information systems

Scaling Techniques (cont.)

Replication/caching – make copies of data available at different machines:

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

Scaling – Challenges

Applying scaling techniques seems easy, except for:

- 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

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

Summary: Scaling Techniques

- *Principles* for good decentralized algorithms
  - No machine has a complete state
  - Make decisions based on local information
  - A single failure does not bring down the system
  - No global clock

- *Techniques*
  - Asynchronous communication
  - Distribution
  - Caching and replication

Challenges and Pitfalls

Many distributed systems are needlessly complex due to 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

References

- Distributed Systems: Principles and Paradigms by Tanenbaum and van Steen, chapter 1