Distributed System

期末考试

- 时间：待定

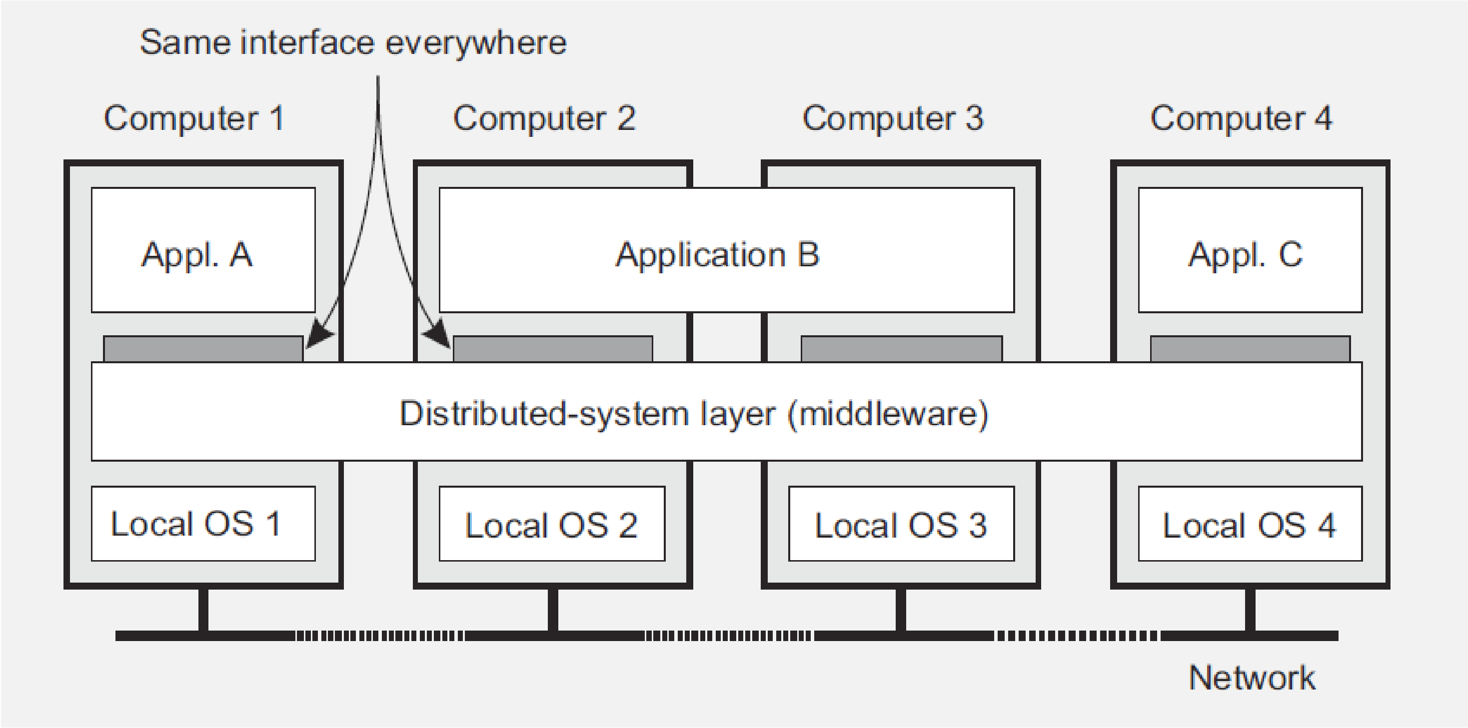
- 闭卷
- 占总评成绩70%

分布式系统模型

- 什么是分布式系统，分布式系统的目标?
- 为什么要分布式?
- 分布式系统透明性和开放性的含义。
- 分布式操作系统、网络操作系统和基于中间件的系统。
- 分布式系统的类型。

Distributed Systems: Definition

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



Goals of Distributed Systems

- Making resources available
- Distribution transparency
- Openness
- Scalability

Types of Distributed Systems

- Distributed computing systems
- Distributed information systems
- Distributed pervasive systems

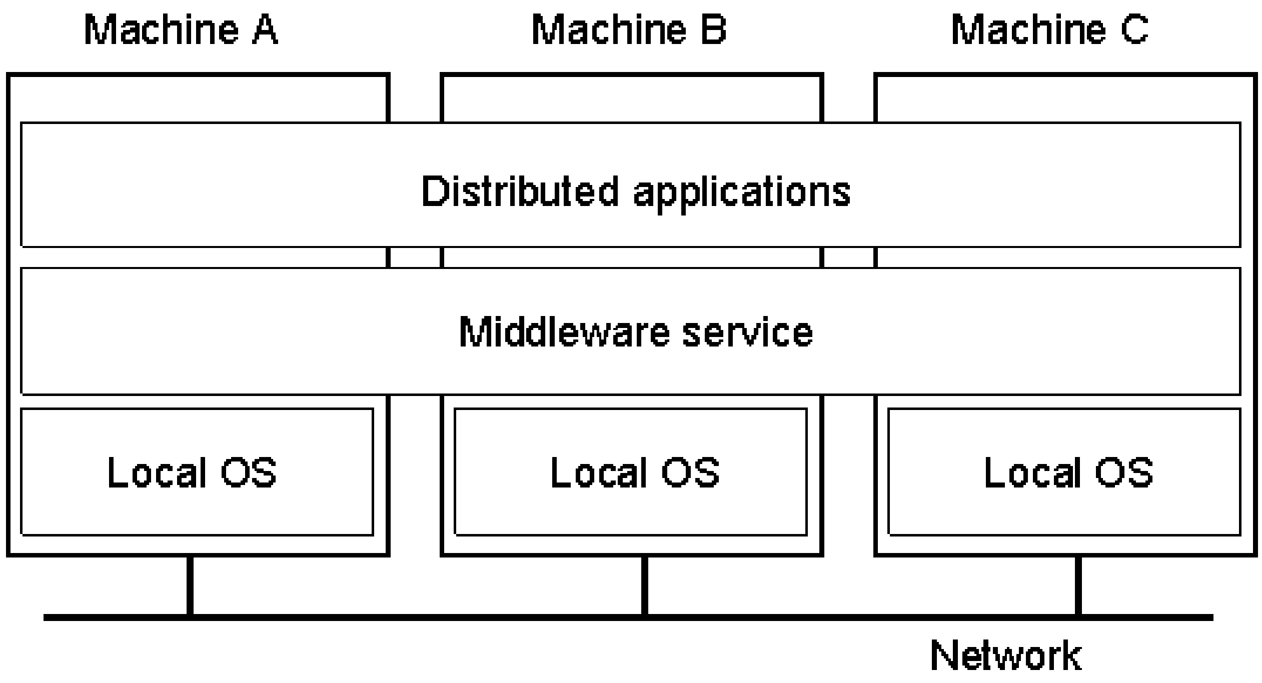
分布式系统架构

- 分布式系统架构风格
- 分布式系统组织形式
- 客户-服务器模式和对等模式
- 分布式系统组织为中间件

Organization

- Centralized
- Decentralized
- Hybrid

A distributed system organized as middleware



进程与线程

- 进程和线程
- 代码迁移
  - 强迁移 vs. 弱迁移

Processes vs. Threads

- A process is different than a thread
- Thread: "Lightweight process" (LWP)
  - An execution stream that shares an address space
  - Multiple threads within a single process
- Example:
  - Two processes examining memory address 0xffe84264 see different values (I.e., different contents)
  - Two threads examining memory address 0xffe84264 see same value (I.e., same contents)

通信

- 通信的类型
- 远程过程调用RPC
  - RPC的工作过程
  - 故障处理
  - 动态绑定
- 基于消息的通信
  - 持久性/非持久性
  - 同步/异步
  - 流数据

同步与资源管理

- 同步问题
- 时钟同步机制
- 逻辑时钟
  - Lamport算法
  - 向量时戳
- 分布式系统中的互斥访问
- 分布式系统中的选举机制

Logical vs Physical Clocks

- Clock synchronization need not be absolute! (due to Lamport, 1978):
  - If two processes do not interact, their clocks need not be synchronized.
  - What matters is not that all processes agree on exactly what time is it, but rather, that they agree on the order in which events occur.
- For algorithms where only internal consistency of clocks matters (not whether clocks are close to real time), we speak of logical clocks.
- For algorithms where clocks must not only be the same, but also must not deviate from real-time, we speak of physical clocks.

复制与一致性

- 复制的优势与不足
- 数据一致性模型
- 数据一致性协议实例
  - 基于法定数量的协议

Replication

- Why replicate?
  - Reliability
    - Avoid single points of failure
  - Performance
    - Scalability in numbers and geographic area
- Why not replicate?
  - Replication transparency
  - Consistency issues
    - Updates are costly
    - Availability *may* suffer if not careful

Summary of Consistency Models

Consistency	Description
Strict	Absolute time ordering of all shared accesses matters.
Linearizability	All processes must see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp
Sequential	All processes see all shared accesses in the same order. Accesses are not ordered in time
Causal	All processes see causally-related shared accesses in the same order.
FIFO	All processes see writes from each other in the order they were used. Writes from different processes may not always be seen in that order

(a)

Consistency	Description
Weak	Shared data can be counted on to be consistent only after a synchronization is done
Release	Shared data are made consistent when a critical region is exited
Entry	Shared data pertaining to a critical region are made consistent when a critical region is entered.

(b)

a) Consistency models not using synchronization operations.

b) Models with synchronization operations.

Client-Centric Consistency

- More relaxed form of consistency → only concerned with replicas being eventually consistent (eventual consistency).
- In the absence of any further updates, all replicas converge to identical copies of each other → only requires guarantees that updates will be propagated.
- Easy if a user always accesses the same replica; problematic if the user accesses different replicas.
  - Client-centric consistency: guarantees for a single client the consistency of access to a data store.

容错

- 可信系统(Dependable System)特征
- 提高系统可信性的途径
- K容错系统
- 拜占庭问题（ Byzantine Problem）
- 系统恢复
  - 回退恢复
  - 前向恢复
- 检查点（Check point）

Distributed commit

- Two-phase commit
- Three-phase commit
- Essential issue
  - Given a computation distributed across a process group, how can we ensure that either all processes commit to the final result, or none of them do (atomicity)?