Sistemas Distribuídos

Summary

Eurico Pedrosa António Rui Borges

Universidade de Aveiro - DETI

2025-05-29

Introduction to Distributed Systems

Course Overview

- Understand principles and practical design of distributed systems
  - Communication and synchronization
  - Java-based implementations
- Develop skills to build distributed applications

Key Learning Outcomes

- Grasp design issues in distributed systems
- Tackle concurrency and consistency in real systems
- Apply Java RMI and message passing

Concurrency

Concurrency

- Computer Architecture
  - CPU
  - Main Memory
  - Pipelining
- Program vs Process
  - Process State
- Multithreading
  - Organization

System Models

Architectural Models

- **Client-Server**: centralized resource control
- **Peer-to-Peer**: equal roles, replication
- **Publisher-Subscriber**: broker decouples producers and consumers

Fundamental Models

- **Interaction**: bandwidth, latency, jitter
- **Failure**: omission, timing, byzantine
- **Security**: process and channel threats, access control

Message Passing and Communication

Communication Fundamentals

- **Latency**, **transfer rate**, **bandwidth**
- **Synchronous** vs **asynchronous** primitives
- **Blocking** vs **non-blocking** operations

Middleware and Sockets

- TCP: reliable, bidirectional, connection-based
- UDP: connectionless, low-overhead
- Socket identified by IP and port

Distributed Execution

Remote Invocation Principles

- Marshaling/unmarshaling of data
- Stub acts as proxy for remote objects
- Communication via structured messages

Client-Server Architecture

- Server base thread + proxy agent
- Mixed architecture: server also acts as client
- Java serialization simplifies implementation

Concurrency in Distributed Systems

General Principles

- **Independent** vs **cooperating** processes
- **Critical regions**: mutual exclusion essential
- **Deadlock** and **indefinite postponement**

Synchronization Devices

- **Monitors**: encapsulated access with wait/signal
- **Semaphores**: general-purpose mutual exclusion
- Java concurrency tools: barriers, locks, atomic ops

Synchronization and Time

Physical and Logical Time

- **Global** vs **local** vs **logical** time
- Synchronization limits due to drift and latency

Synchronization Techniques

- Cristian's Method (UTC server)
- Berkeley Algorithm (internal synchronization)
- NTP: hierarchical, resilient synchronization over Internet

Logical Clocks

Lamport Clocks

- Capture **happened-before** relation
- Scalar timestamps ensure partial ordering
- Extended timestamps for total ordering

Vector Clocks

- Precise causality tracking
- Each process maintains vector of logical times
- Allows detecting concurrency vs causality

Group Communication

Mutual Exclusion Protocols

- **Centralized**: guardian manages access
- **Token Ring**: token grants access in sequence
- **Ricart & Agrawala**: logical clocks for total order
- **Maekawa**: voting subsets to minimize messages

Leader Election

- **Ring-based** and **unstructured group** algorithms
- Key properties: termination, unambiguity, consensus
- Failure handling: timeout, retries, ACK-based recovery

Consistency and Replication

Models of Consistency

- **Strict Consistency**: ideal but unrealistic
- **Linearizability**: respects real-time order
- **Sequential Consistency**: respects program order
- **Causal Consistency**: enforces causal relations
- **FIFO Consistency**: per-process order preserved

Implementation Insights

- Use of logical clocks (scalar or vector)
- Propagation of write-like operations
- Application-dependent trade-offs

Distributed Transactions

ACID Transactions

- **Atomicity**, **Consistency**, **Isolation**, **Durability**
- Local commit or global abort
- Locking to prevent race conditions

Commit Protocols

- **2-Phase Commit (2PC)**:
  - Voting $\longrightarrow$ Decision (commit/abort)
- **3-Phase Commit (3PC)**:
  - Adds Pre-commit phase
  - Ensures progress and non-blocking behavior

Nested Transactions

- Hierarchical coordination of sub-transactions
- Abort propagates to all sublevels