Contents

Contributors:
- Daniel Fitz (Sanchez)

0.1 Assessment

- Final Exam (50%)
  - All goals tested except for developing applications
  - **Closed book exam**
- Two assignments
  - Individual programming assignment (25%)
    Implementation of a context-aware distributed application using RMI and “publish/subscribe” (event based architecture)
  - Group research/written assignment (25%)
    Group assignment on functionality and design issues of various distributed systems (e.g. grid computing, cloud computing, pervasive computing)

Chapter 1

Lecture Notes

1.1 Introduction to Distributed Systems

1.1.1 Definitions of Distributed Systems

> A collection of independent computers that appear to its users as a single coherent system (Andrew Tannenbaum)

> A system where I can't get my work done because a computer has failed that I've never even heard of (Leslie Lamport)

A distributed system is a collection of independent computers that are used jointly to perform a single task or to provide a single service.

Note 1: Characteristics

- Multiple computers
  CPU, memory, storage, I/O
- Interconnections
  variety of interconnection architectures
- Resources
  remote access to resources
  resource can be shared

1.1.2 Goals of Distributed Systems

- Transparency (hiding distribution)
  System presents itself as a single computer system
- Openness
  Interoperability, portability, heterogeneity
- Scalability
  Ability to grow

Transparency

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 is replicated
Concurrency: Hide that a resource may be shared by several competitive users
Failure: Hide the failure and recovery of a resource

Openness

- Interoperability
- Portability
- Heterogeneity
- Standard interfaces
- Interface Definition Language (IDL)

Scalability

' Three axis of scalability:
- Administratively
- Geographically
- Size (users, resources)

Algorithms vs Scalability Decentralized algorithms should be used:
- No machine has complete information about the system state
- Machines make decisions based only on local information
- Failure of one machine does not ruin the algorithm
- There is no implicit assumption that a global clock exists

Scaling Techniques
- Hiding communication latencies
  Asynchronous communication
  Client-side processing
- Distribution
  Split and spread functionality across the system
  Decentralize algorithms
- Replication (including caching)

If asynchronous communication cannot be used - communication should be reduced

1.1.3 Types of Distributed Systems

Distributed Computing Systems

- Cluster Computing Systems

  Just a bunch of computers all connected over a shared network
- Grid Computing Systems

  Layered System: Applications $\rightarrow$ Collective Layer $\rightarrow$ (Connectivity layer / Resource layer) $\rightarrow$ Fabric layer
- Cloud Computing

  Paradigm for enabling **network access** to a scalable and elastic pool of **shareable physical or virtual resources** with on-demand self-service provisioning and administration

Distributed Information Systems

- Transaction processing systems

  There are many information systems in which many distributed operations on (possibly distributed) data have to have the following behavior (either all of the operations are executed, or none of them is executed):

  BEGIN_TRANSACTION: Mark the start of a transaction

  END_TRANSACTION: Terminate the transaction and try to commit

  ABORT_TRANSACTION: Kill the transaction and restore the old values

  READ: Read data from a file, a table, or otherwise

  WRITE: Write data to a file, a table, or otherwise

Note 2: Distributed Transactions - Model

A transaction is a collection of operations that satisfies the following ACID properties:

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 transaction. This does not exclude the possibility of invalid, intermediate states during the transaction's execution.

Isolation (Serialisability): Concurrent transaction 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.

- Enterprise application integration

  Middleware as a communication facilitator in enterprise application integration

  Multiple applications communicate to the middleware which then talks to all the server-side applications

Distributed Pervasive Systems

Pervasive systems:

- Embedded devices
- Mobile devices
- Heterogeneous networks
- (Autonomic) Adaptation to context changes

  Adaptation to changes in the infrastructure

  Adaptation to user tasks/needs

Requirements for pervasive systems:

- Embrace contextual changes
- Encourage ad hoc composition
- Recognize sharing as the default

Home Systems (Smart Homes) Integration of entertainment and appliances into an "intelligent" adaptive system. May include health-monitoring and also provide support for independent living of the elderly.

Sensor Networks There is a variety of sensor networks, e.g.

- A small set of sensors supporting smart home

- A network of thousands of sensors providing climate monitoring

1.2 Architectures of Distributed Systems

Architecture styles

- Layered architectures
- Object-based architectures
- Data-centered architectures
- Event-based architectures

System architectures

(how software components are distributed on machines)

- Centralized architectures (client-server: two-tiered, three-tiered, N-tiered)
  - The simplest organization is to have only two types of machines:
    - A client machine containing only the programs implementing (part of) the user-interface level
    - A server machine containing the rest

      the programs implementing the processing and data level
- Decentralized architectures (peer-to-peer)
  - Overlay network is constructed in a random way
  - Each node has a list of members but the list is created in unstructured (random) way
- Hybrid architectures (edge-server, collaborative DS)

  Clients participate in providing services: e.g. file sharing, when part of file is downloaded it's seeded to other clients

Note 3: Application Layering

- The user-interface level
- The processing level
- The data level

1.2.1 Adaptability and self-management in DS

- Role of middleware is to provide some degree of distribution transparency

  Hiding distribution of data, processing and control
- Middleware may have a particular architectural style, e.g.

  Object-based (CORBA)

  Event-based (most of middleware built for adaptive, context-aware applications)
- Middleware should be adaptive to meet requirements of various applications

Using interceptors to adapt control

- Interceptors change flow of control and allow additional code to be executed
- Many object-based DS use interceptors to change flow of control in the object invocation

  Requests (object invocations) can be intercepted

  Messages can be intercepted

General Approaches to Adaptive Software

- **Separation of concerns**

  e.g. *aspect oriented programming* (not very successful)
- **Computational reflection**

  e.g. *reflective middleware*
- **Component-based design**

  Adaptation through composition

  Statically at design time

  Dynamically at run time (requires support for late binding)

Self-management / autonomic computing

- Many distributed systems are complex and require self-management and behaviour adaptation
- Autonomic computing (or *self-**)
  - Self-configuration
  - Self-healing
  - Self-management
  - Context-awareness, etc
- Adaptation takes place by one or more feedback control loops

The Feedback Control Model

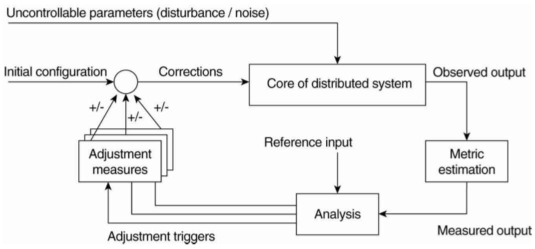

Figure 1.1: The logical organisation of a feedback control system

1.2.2 Processes

Note 4: Core OS functionality

- Process Manager
  Process lifecycle
- Thread Manager
  Thread lifecycle
- Communication Manager
  Communication between threads attached to different process
- Memory Manager
  Physical/virtual
- Supervisor
  Dispatching interrupts, traps, ...

Definitions

- **Process**
  - Program being executed
  - Consists of address space and execution state (program counter, stack pointer, processor status word, contents of registers and system-call state)
- **Thread** (of execution)
  - Abstraction of activity within process

Thread has less overhead and so interruptions and switching can be faster. Threads also have shared memory

The use of threads in DS

- Starting thread to handle incoming request much cheaper than starting new process
  - single-threaded server prevents simple scaling to multiprocessor
  - as with client: hide network latency by reacting to next request while previous being transmitted
- Better structure than nonblocking I/O
  - single thread, blocking I/O doesn't scale up
  - simple blocking calls simpler overall structure

Note 5: Multithreaded Servers

Threads: Parallelism, blocking system calls
Single-threaded process: No parallelism, blocking system calls
Finite-state machine: Parallelism, nonblocking system calls

Virtualisation in DS

Interface at different levels:

1. An interface between the hardware and software consisting of machine instructions
   that can be invoked by any program
2. An interface between the hardware and software, consisting of machine instructions
   that can be invoked only by privileged programs, such as an operating system
3. An interface consisting of system calls as offered by an operating system
4. An interface consisting of library calls
   generally forming what is known as an application programming interface (API)
   in many cases, system calls are hidden by an API

Clients/Servers

Note 6: Clients

- Clients are often **thin**
  Large portion of processing required to support sophisticated user interfaces can by provided by servers
- Clients should support transparencies
  communication hiding
  access transparency (client stub)
  location, migration, relocation transparencies

Servers General Design Issues
Stateful server: maintains information on clients
hard to handle failures
Stateless server: knows nothing about clients
can change own state without informing clients

Chapter 2

Tutorials

2.1 Introduction to Distributed Systems

What is the role of middleware in a distributed system? Middleware provides distributed transparency such as:

- Access Transparency
- Location Transparency
- Concurrency Transparency

Explain what is meant by distribution transparency and give examples of different types of transparency. Distribution transparency allows aspects of distributed systems (such as accessing of data or individual software components) to be hidden and appear to the end user as a single system. Examples:

- Access Transparency
- Location Transparency
- Migration Transparency
- Relocation Transparency
- Replication Transparency
- Concurrency Transparency
- Failure Transparency

Why is it sometimes so hard to hide the occurrence and recovery from failures in a distributed system? It can be difficult to identify the state of remote components. For example how do you tell the difference between unavailable resource and a slow resource?

Why is it not always a good idea to aim at implementing the highest degree of transparency possible? Aiming at the highest degree of transparency may lead to a considerable loss of performance that users are not willing to accept.

What is an open distributed system and what benefits does openness provide? An open distributed system offers services according to a clearly defined set of rules and interfaces. An open system is capable of easily interoperating with other open systems but also allows applications to be easily ported between different implementations of the same system. Furthermore, an open distributed system allows software/hardware components of different natures (e.g. Windows, Linux workstations etc) to participate in the system. A few benefits are:

- The same system can deliver the service to different types of clients and applications
- The same system can work given different computing environments
- The same system can be extended to allow computing and storage technologies by different vendors

Describe precisely what is meant by a scalable system. A system is scalable with respect to either its number of components, geographical size, or number and size of administrative domains, if it can grow in one or more of these dimensions without an unacceptable loss of performance.

Scalability can be achieved applying different techniques. What are these techniques? Scaling can be achieved through:

- Workload and data distribution. This requires distributed/parallel algorithms
- Using decentralised architecture
- Data replication, and caching

2.1.1 Architecture of Distributed Systems

If a client and a server are placed far apart, we may see network latency dominating overall performance. How can we tackle this problem?

1. Buffering the communication so there is enough data presented to the client while more data is being transferred
2. Perform more client-size processing and caching to reduce communication load
3. Multithreaded communication requests on client and server to reduce network latency

What is a three-tiered client-server architecture? A three-tiered client-server architecture consists of three logical layers, where each layer is, in principle, implemented at a separate machine.

The highest layer consists of a client user interface, the middle layer contains the actual application, and the lowest layer implements the data that are being used.

What is the different between a vertical distribution and a horizontal distribution?

Vertical distribution: Multiple layers, each implemented on a different machine

Horizontal distribution: Single layer, implemented across multiple machines (distributed database)

In a structured overlay network, messages are routed according to the topology of the overlay. What is an important disadvantage of this approach? When a message is routed across a structure overlay network (which is a logical network) the shortest path between source and destination may not be the physical shortest path. While the source and receivers may be logically very close to each other, they could be physically at the remotest part of the network.