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 → Collective Layer → (Connectivity layer / Resource layer) → 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

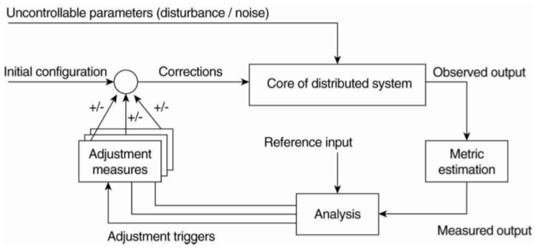

Figure 1.1: The logical organisation of a feedback control system

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

1.3 Communication

- Interprocess communication in DS is based on message exchange
- Low level (transport) message passing does not provide distribution transparency
  higher level models are needed
- Distributed application need a variety of communication semantics
  - Remote Procedure Call (RPC)
  - Message-Oriented Middleware (MOM)
  - Data streaming

1.3.1 Layered Protocols

Layers, interfaces, and protocols in the **Open Systems Interconnection Reference Model** (OSI)
Application: Application protocol
Presentation: Presentation protocol
Session: Session protocol
Transport: Transport protocol
Network: Network protocol
Data link: Data link protocol
Physical: Physical protocol

Middleware Layer

Provides common protocols that can be used by many different applications

- Security protocols
- Transaction protocols
- High-level communication models (e.g. RPC, message queuing services, etc)

1.3.2 Types of Communication

Synchronicity in Communication

> **Persistent communication** - message is stored in the system until receiver becomes active

Persistent asynchronous communication. A will send a message to B and not wait for response back

Persistent synchronous communication. A sends a message and wait's for B to respond before continuing execution

Transient asynchronous communication. A sends a message, B will only receive the message if it is running

Receipt-based transient synchronous communication. A sends a message, B receives the message if it is active and returns an acknowledgement

Delivery-based transient synchronous communication at message delivery. A sends a message and waits for the acknowledge from B, B will receive the message and send an acknowledge when it starts on the job

Response-based transient synchronous communication. A sends a message and waits for the acknowledge from B, B will receive the message and send an acknowledge when it has finished the job

Steps of a Remote Procedure Call

1. Client procedure calls client stub (procedure invocation)
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

Note 7: Parameter Passing

Parameter marshalling: There's more than just wrapping parameters into a message. Client and Server:

- have different data representations
- have to agree on the same encoding (basic/complex data values)
- Interpret data and transform them into machine-dependent representation

1.3.3 Parameter Specification and Stub Generation

- Parameters passed by value do not pose problems
- Parameters passed by reference - some partial solutions exist (e.g. a small array can be sent to the server)

- Interfaces (procedures) are often specified in Interface Definition Language (IDL) and compiled into stubs

Writing a Client and a Server

Three files output by the IDL compiler:
- A header file (e.g. interface.h, in C terms)
- The client stub
- The server stub

1.3.4 Binding a Client to a Server

- Registration of a server makes it possible for a client to locate the server and bind to it
- Server location is done in two steps:
  1. Locate the server's machine
  2. Locate the server on that machine

1.3.5 Message-Oriented communication

- RPC (and RMI) are **synchronous** and **transient** (both sender and receiver have to be active
- Transport level communication like TCP and UDP also **require that the sender and receiver are active** (transient communication)
- Some applications require that messages are kept in the system until the receiver becomes active (e-mail is one example, but there are many more applications of this type)

Asynchronous Communication

- MOM – Message-Oriented Middleware
- Can store messages in the system
- Applications communicate by inserting messages in queues
- Messages are delivered by an overlay network of application layer servers (routers)
- Each application has its own private queue to which other applications can send messages
- There is no guarantee on when the message will be delivered and that it will be read

Message-Queueing Model

Put: Append a message to a specific queue – non-blocking

Get: Block until the specified queue is nonempty, and remove the first message

Poll: Check a specified queue for messages, and remove the first. Never block

Notify: Install a handler to be called when a message is put into the specified queue

Message brokers

- Main role of brokers is to transform messages from sender's format to receiver's format
- This functionality can be generalised to brokers matching applications based on messages

  Applications *publish* messages

  Applications *subscribe* for messages

Note 8: MCA: Message Channel Agent

Responsible for checking a message, wrapping it, and sending it to associated receiving ends

Channels

Transport type: Determines the transport protocol to be used

FIFO delivery: Indicates that messages are to be delivered in the order they are sent

Message length: Maximum length of a single message

Setup retry count: Specifies maximum number of retries to start up the remote MCA

Delivery retries: Maximum times MCA will try to put received message into queue

Message Transfer

MQopen: Open a (possibly remote) queue
MQclose: Close a queue
MQput: Put a message into an opened queue
MQget: Get a message from a (local) queue

1.3.6 Transient Communication

Berkeley Sockets

Socket: Create a new communication end point
Bind: Attach a local address to a socket
Listen: Announce willingness to accept connections
Accept: Block caller until a connection request arrives
Connect: Actively attempt to establish a connection
Send: Send some data over the connection

Receive: Receive some data over the connection
Close: Release the connection

1.3.7 Actor model for communication

- The actor model adopts the philosophy that *everything is an actor*
- Can respond to a message it receives, can send a finite number of messages to other actors
- Enabling asynchronous communication and control structures as patterns of passing messages
- Resemble the Enterprise application integration framework

1.3.8 Stream-oriented communication

- Some applications need to exchange time-dependent information e.g. video, audio (continuous media)
- Previously discussed models do not consider time
- In continuous media the temporal relationship between different data items is essential

1.3.9 Different transmission modes

- Asynchronous transmission mode
  Sequential transmission without restrictions on when data is to be delivered (e.g. file transfer)
- Synchronous transmission mode
  Max end-to-end delay for each unit in a data stream (e.g. sensor sample temperature)
- Isochronous transmission mode (streams)
  Data transfer bounded by maximum and minimum end-to-end delay (bounded jitter e.g. audio/video)

1.3.10 Streams

Streams can be simple or complex (i.e. can include several related substreams)

- Unidirectional (source $\rightarrow$ sinks)
- Simple (single flow of data e.g. audio, video)
- Complex (stereo audio, movie)

Streams and QoS

Streams need timely delivery of data

- Non functional requirements (time, bandwidth, volume, reliability) are expressed as Quality of Service (QoS)
- There are many models for QoS specifications: e.g. IntServ, DiffServ, MPLS
- In IntServ (Integrated Services), QoS is specified as a flow specification – flow reservation
- In DiffServ (Differentiated Services), QoS is specified for a class (e.g. expedited forwarding class)

Properties for Quality of Service:

- The required bit rate at which data should be transported (application specific)
- The maximum delay until a session has been set up (when to start sending data)
- The maximum end-to-end delay
- The maximum delay variance, or jitter
- The maximum round-trip delay

Stream Synchronisation

- Given a complex stream, how are substreams synchronised?
- Synchronisation takes place at the level of data units (e.g. synchronise two streams only between data units)
- Issues which need to be considered:
  Mechanisms for synchronising streams
  Distribution of those mechanisms

Distribution of synchronisation mechanisms

- The receiving side needs to have a complete synchronisation specification
- Synchronisation specification can be multiplexed together with other substreams into a complex stream (and is demultiplexed after receiving)

Multicast communication

- Application level multicasting – setting a path for information dissemination
  - Nodes (applications) organise into an overlay network
  - Overlay network disseminates information
  - Network layer routing is independent of the overlay (communication may not be optimal)
- Overlay organisation
  - Nodes may organise themselves into a tree, or

- Nodes organise themselves into a mesh network

Gossip-based data dissemination

- Epidemic protocols are often used for data dissemination
  - There is no central component which co-ordinates dissemination
  - Information is propagated using local information only
- Node is *infected* if is has data which it wants to spread
- Node which has not seen this data is *susceptible*

Note 9: Information Dissemination Models

- Anti-entropy propagation model
  - Node P picks another node Q at random
  - Subsequently exchanges updates with Q
- Approaches to exchanging updates
  - P only pushes its own updates to Q
  - P only pulls in new updates from Q
  - P and Q send updates to each other
- One variant is the "gossiping" protocol

1.4 Naming

1.4.1 Names, Identifiers and Addresses

- Names refer to entities
  e.g. processes, users, mailboxes, network connections
- Special types of names exist:
  - Addresses
  - Identifiers
  - Human-friendly names

Addresses

- To use (operate on) an entity you need an access point
- The name of an access point is an *address*
- Entities can have more than one access point at a time
- An entity may change access points e.g. when it changes location
- *Location-independent names* are not tied to an address

Identifiers

Special type of name that:
- Refers to at most one entity
- Always refers to the same entity
- Are limited to one per entity

Human-friendly names

Generally represented as a character string e.g. `www.news.com.au` or `/root/ryan/slides`

1.4.2 Flat Naming

Random bit strings (name doesn't help locate access point)
- Names must be resolved to address
- There are many name resolution approaches for flat names:
  - Broadcasting (or multicasting) approaches
  - Forwarding pointers
  - Home-based approaches (e.g. Mobile IP)
  - Distributed Hash Tables (DHTs)
  - Hierarchical approaches

Broadcasting

Broadcasting (or multicasting) the ID requesting the entity to return its current address. e.g. ARP (Address Resolution Protocol)

Forwarding Pointers

- When entity moves, it leaves a pointer to its new location
- To find entity, must follow trail of pointers to entity's current location
- Problems:
  - Can get long chains of pointers (not scalable)
  - If chain breaks, can't find entity
- Potential solution:
  Stub Scion Pair (SSP) chains

Home-Based Approach

- Approach designed for mobile entities

- Entities have home location that keeps track of entity's current location
- Example of home-based approach is Mobile IP:
  - Mobile entity has fixed IP address
  - All traffic to mobile entity goes to entity's Home Agent (home location)
  - Home Agent forwards traffic to Care-Of-Address (mobile entity's IP in its current network)
  - Whenever mobile entity changes network, it gets new Care-Of-Address, which it registers with Home Agent

Distributed Hash Tables

- Consist of many distributed nodes
- Map data to a key value using a hash function
- Each node is responsible for key values (and the associated data) in a particular range
- Have fast lookup times

Hierarchical Approach

- Network is divided into **domains**, with each domain subdivided into smaller subdomains
- **Leaf domain** is lowest level domain (typically LAN or Cell in mobile phone network)
- Each domain has a **directory node** that keeps track of entities in that domain
- Directory nodes for leaf domains store address of entities in that domain
- Directory nodes for non-leaf domains store reference to lower-level domain containing entities

1.4.3 Structured Naming

- Flat names are not convenient for humans
- Structured names:
  - Composed from simple human-readable names
  - Generally supported by naming systems
  - Names are organised into a **name space**

Name Spaces

- Name spaces for structured names are represented as a labelled, directed naming graph in which:
  - A **leaf node** represents a (named) entity
  - A **directory node** is an entity that refers to other nodes

    Outgoing edge is represented as (edge label, node identifier)
  - **Root node** has only outgoing edges
- Naming graphs are usually directed acyclic graphs
- Each path in a naming graph can be referred to by a sequence of edge labels separated by a special character

Name Resolution in a Name Space

- Need to know how and where to start → need closure mechanism
- Closure mechanism deals with selecting initial node in a name space from which name resolution is to start
- Closure mechanisms are often implicit

Linking and Mounting

- Aliases
  - Another name for the same entity
  - Symbolic links in UNIX enable more than one path for a file
- Mounting
  - Used to merge different name spaces transparently

Name Resolution

- Structured names resolved using naming service (implemented by name servers)
- Naming service allows addition, removal and lookup of names
- DS naming services are distributed
- Large-scale distributed naming services are usually hierarchical:

  Global level: high-level directory nodes

  Administrational level: mid-level directory nodes grouped into separate administrations

  Managerial level: low-level directory nodes within a single administration
- The Domain Name System is a good example of a structured name resolution mechanism

DNS

- Large distributed name service used by Internet
- DNS name space is hierarchical

- DNS labels use alphanumeric character strings separated by a "."
- A path name in DNS is called a domain name
- DNS is primarily used to lookup IP addresses for hosts and mail servers
- Client (e.g. browser) contacts name resolver
- Name resolver uses either **iterative** or **recursive** approach
- Iterative name resolution:
  1. Name resolver contacts name servers for help
  2. Starts at root name server
  3. Each name server resolves as much of name as it can before referring name resolver to another name server who knows more
  4. Process repeats until name is fully resolved, or cannot be resolved anymore
- Recursive name resolution:
  1. Name resolver sends name to root name server
  2. Intermediate result not passed to client, rather it is sent to next name server
  3. Process repeats until name is fully resolved, then servers pass fully resolved name back
  4. Root name server passes fully resolved name to name resolver

Comparison between Iterative and Recursive:

- Recursive approach has higher performance demand on name servers
- Recursive approach can use result caching more effectively
- Iterative approach has higher communication costs

1.4.4 Attribute-based Naming

- Each entity is described by *(attribute, value)* pairs
- Attribute-based queries:

  Users specify attributes they are looking for

  Naming system should return one (or more) entities with the specified attributes
- Attribute-based naming systems are commonly known as **directory services**
- Lightweight Directory Access Protocol (LDAP) is a common directory service

LDAP

- Derived from OSI X.500 directory service
- LDAP directory service stores *directory entries*
- Directory entries consist of (attribute, value) pairs
- Directory Information Base (DIB) is collection of all directory entries in an LDAP service
- Each LDAP directory entry has globally unique name (Directory Information Tree) based on hierarchy of naming attributes

  e.g. /C=NL/O=Vrije/OU=Comp.Sc

Table 1.1: Simple Example of LDAP directory entry

Attribute	**Abbr**	**Value**
Country	C	NL
Locality	L	Amsterdam
Organisation	O	Vrije
OrganisationalUnit	OU	Comp.Sc
CommonName	CN	Main Server
Mail_Servers	–	137.37.20.3, 130.37.24.6
FTP_Server	–	130.37.20.20
WWW_Server	–	130.37.20.20

Decentralised Schemes

- Driven by advent of peer-to-peer
- Need efficient mapping of (attribute,value) pairs to avoid exhaustive search of networks
- Two Approaches:
  - Distributed Hash Tables (INS/Twine)
  - Semantic Overlay Networks

Note 10: Distributed Hash Tables

- In INS/TWINE each entity (resource) is described by a hierarchical attribute-value tree (AVTree)
- Every path from root of AVTree gets unique hash value
- Node in DHT responsible for hash value will keep reference to actual resource
- Query for (type-book) will get hashed to value 5 and sent to node responsible for storing hash value 5

Note 11: Semantic Overlay Networks

- When there is no organised attribute-based naming resolution scheme, nodes must discover for themselves where resources are located
- To make queries efficient, nodes can track nodes with similar resources
- Measuring similarity based on attributes is difficult $\rightarrow$ different nodes have different definitions of attributes
- Possibly ignore attributes and use file names

  Similarity measured as number of files in common

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.